CLIMADA documentation

Release 6.1.0

CLIMADA contributors

Sep 30, 2025

CONTENTS

CLIMADA (CLIMate ADAptation) CLIMADA is a free and open-source software framework for climate risk assessment and adaptation option appraisal. Designed by a large scientific community, it helps researchers, policymakers, and businesses analyse the impacts of natural hazards and explore adaptation strategies.

CLIMADA is primarily developed and maintained by the Weather and Climate Risks Group at ETH Zürich.

If you use CLIMADA for your own scientific work, please reference the appropriate publications according to the *Citation Guide*.

This is the documentation of the CLIMADA core module which contains all functionalities necessary for performing climate risk analysis and appraisal of adaptation options. Modules for generating different types of hazards and other specialized applications can be found in the CLIMADA Petals module.

Useful links: WCR Group | CLIMADA Petals | CLIMADA website | Mailing list

Getting Started Getting started with CLIMADA: How to install? What are the basic concepts and functionalities?

Getting started

User Guide Want to go more in depth? Check out the User guide. It contains detailed tutorials on the different concepts, modules and possible usage of CLIMADA.

To the user guide!

Implementation API reference The reference guide contains a detailed description of the CLIMADA API. The API reference describes each module, class, methods and functions.

To the reference guide!

Developer guide Saw a typo in the documentation? Want to improve existing functionalities? Want to extend them? The contributing guidelines will guide you through the process of improving CLIMADA.

To the development guide!

Hint

ReadTheDocs hosts multiple versions of this documentation. Use the drop-down menu on the bottom left to switch versions. `stable` refers to the most recent release, whereas `latest` refers to the latest development version.

Date: Sep 30, 2025 **Version**: 6.1.0

Copyright Notice

Copyright (C) 2017 ETH Zurich, CLIMADA contributors listed in *AUTHORS.md*.

CLIMADA is free software: you can redistribute it and/or modify it under the terms of the GNU General Public License as published by the Free Software Foundation, version 3.

CLIMADA is distributed in the hope that it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the GNU General Public License for more details.

You should have received a copy of the GNU General Public License along with CLIMADA. If not, see https://www.gnu.org/licenses/.

CHAPTER

ONE

GETTING STARTED

1.1 Quick Installation

The simple CLIMADA installation only requires the *mamba* (or *conda*) python environment manager (look *here*).

If you are already working with mamba or conda, you can install CLIMADA by executing the following line in the terminal:

mamba create -n climada_env -c conda-forge climada

Each time you will want to work with CLIMADA, simply activate the environment:

```
mamba activate climada_env
```

You are good to go!

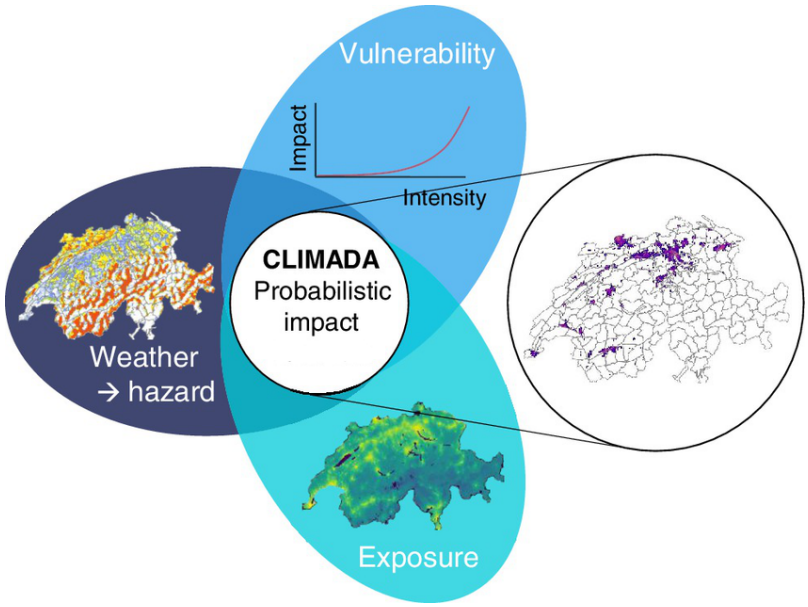
See also

You don't have mamba or conda installed, or you are looking for advanced installation instructions? Look up our *detailed instructions* on CLIMADA installation.

1.2 CLIMADA in a Nutshell

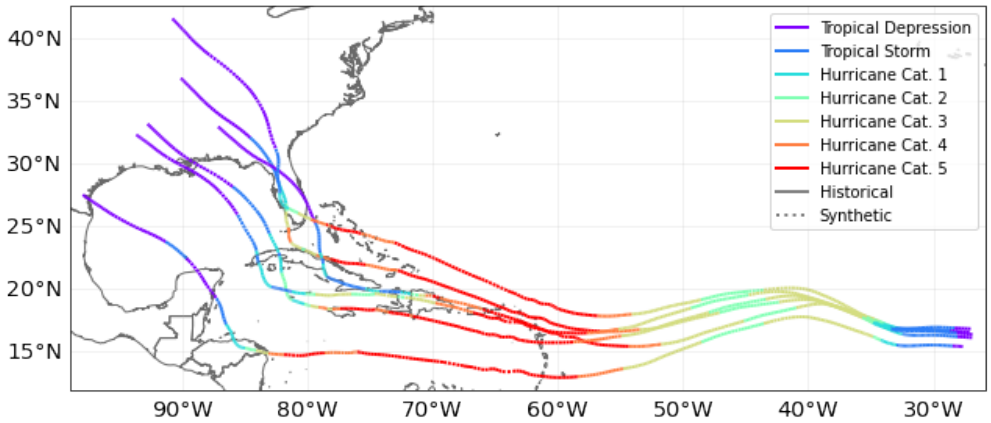
How does CLIMADA compute impacts ?

CLIMADA follows the IPCC risk framework to compute impacts by combining hazard intensity, exposure, and vulnerability. It models hazards intensity (e.g., tropical cyclones, floods) using historical event sets or stochastic simulations, overlaying them with spatial exposure data (e.g., population, infrastructure), and applies vulnerability functions that estimate damage given the hazard intensity. By aggregating these results, CLIMADA calculates expected impacts, such as economic losses or affected populations. See the dedicated *impact tutorial* for more information.



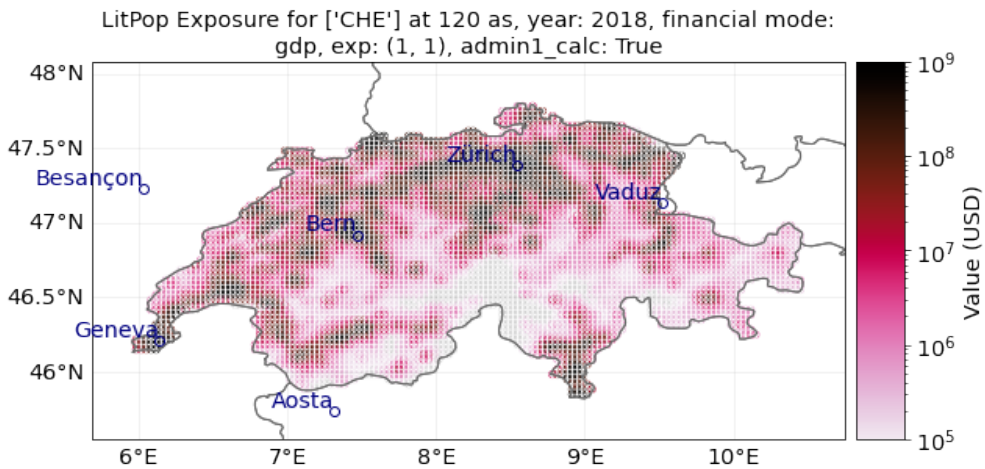
How do you create a Hazard ?

From a risk perspective, the interesting aspect of a natural hazard is its location and intensity. For such, CLIMADA allows you to load your own *hazard* data or to directly define it in the platform. As an example, users can easily load historical tropical cyclone tracks (IBTracks) and apply stochastic methods to generate a larger ensemble of tracks from the historical ones, from which they can easily compute the maximal windspeed, the hazard intensity.



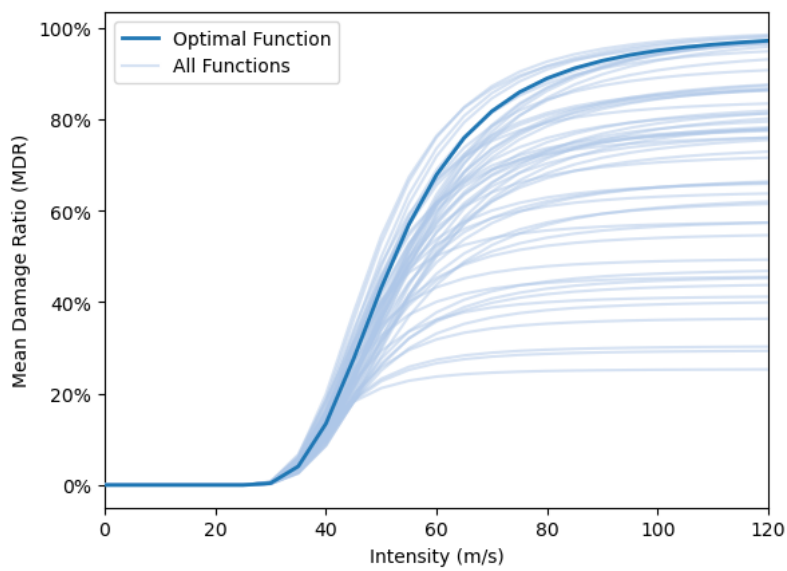
How do we define an exposure ?

Exposure is defined as the entity that could potentially be damaged by a hazard: it can be people, infrastructures, assests, ecosystems or more. A CLIMADA user is given the option to load its own exposure data into the platform, or to use CLIMADA to define it. One common way of defining assets' exposure is through *LitPop*. LitPop dissagrate a financial index, as the country GDP for instance, to a much finer resolution proportionally to population density and nighlight intensity.



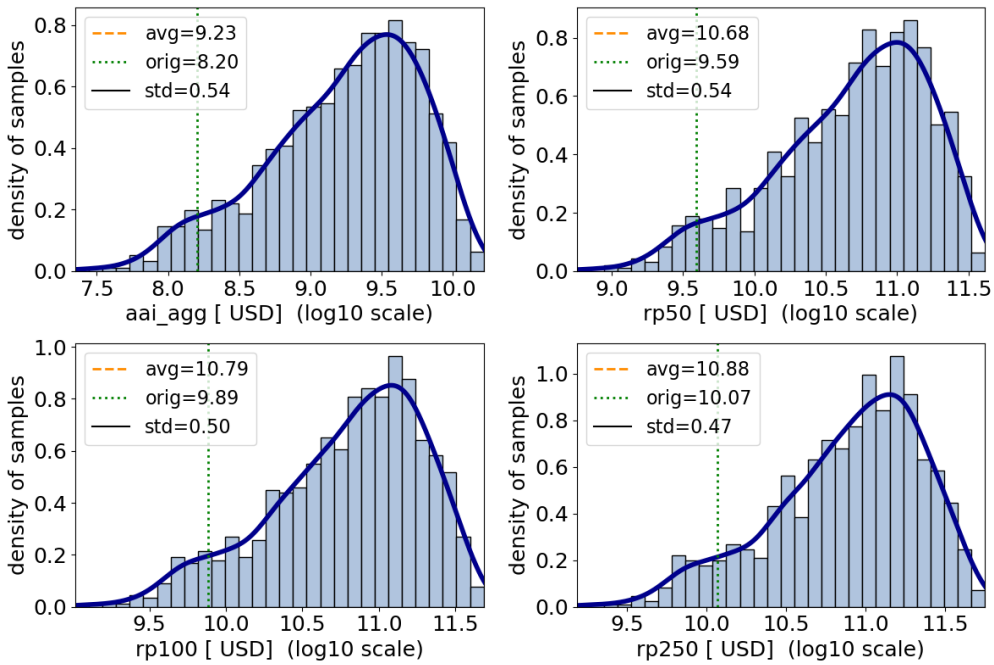
How do we model vulnerability ?

Vulnerability curves, also known as impact functions, tie the link between hazard intensity and damage. CLIMADA offers built-in sigmoidal or step-wise vulnerability curves, and allows you to calibrate your own impact functions with damage and hazard data through the *calibration module*.



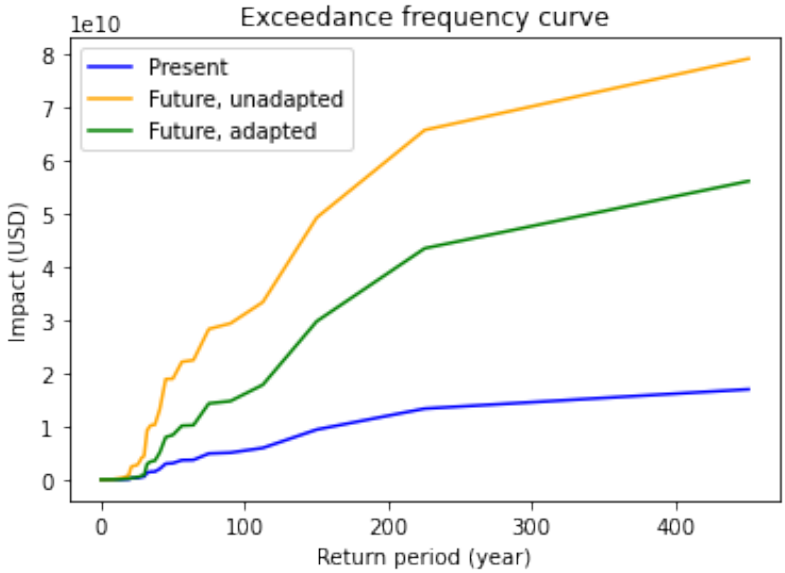
Do you want to quantify uncertainties ?

CLIMADA provides a dedicated module *unsequa* for conducting uncertainty and sensitivity analyses. This module allows you to define a range of input parameters and evaluate their influence on the output, helping you quantify the sensitivity of the modeling chain as well as the uncertainties in your results.



Compare adaptation measures and assess their cost-effectiveness

Is there an adaptation measure that will decrease the impact? Does the cost needed to implement such measure outweight the gains? All these questions can be asnwered using the *cost-benefit* and *adaptation module*. With this module, users can define and compare adaptation measures to establish their cost-effectiveness.



1.2.1 How to navigate this documentation

This page is a short summary of the different sections and guides here, to help you find the information that you need to get started.

Each top section has its own landing page, presenting the essential elements in brief, and often several subsections, which go more into details.

Getting started

The *Getting started* section, where you are currently, presents the very basics of climada.

For instance, to start learning about CLIMADA, you can have a look at the *introduction*.

You can also have a look at the paper repository to get an overview of research projects conducted with CLIMADA.

Programming in Python

It is best to have some basic knowledge of Python programming before starting with CLIMADA. But if you need a quick introduction or reminder, have a look at the short *Python Tutorial*. Also have a look at the python *Python Dos and Don't* guide and at the *Python Performance Guide* for best practice tips.

Tutorials

A good way to start using CLIMADA is to have a look at the tutorials in the *User Guide*. The *10 minute climada* tutorial will give you a quick introduction to CLIMADA, with a brief example on how to calculate you first impacts, as well as your first appraisal of adaptation options, while the *Overview* will present the whole structure of CLIMADA more in depth. You can then look at the specific tutorials for each module (for example if you are interested in a specific hazard, like *Tropical Cyclones*, or in learning to *estimate the value of asset exposure*,…).

Contributing

If you would like to participate in the development of CLIMADA, carefully read the *Developer Guide*. Here you will find how to set up an environment to develop new features for CLIMADA, the workflow and rules to follow to make sure you can implement a valuable contribution!

API Reference

The *API reference* presents the documentation of the internal modules, classes, methods and function of CLIMADA.

Changelog

In the Changelog section, you can have a look at all the changes made between the different versions of CLIMADA

External links

The top bar of this website also link to the documentation of Climada Petals, the webpage of the Weather and Climate Risk group at ETH, and the official CLIMADA website.

Other Questions

If you cannot find you answer in the other guides provided here, you can open an issue for somebody to help you.

1.2.2 Introduction

CLIMADA implements a fully probabilistic risk assessment model. According to the IPCC [*1*], natural risks emerge through the interplay of climate and weather-related hazards, the exposure of goods or people to this hazard, and the specific vulnerability of exposed people, infrastructure and environment.

The unit of measurement for risk in CLIMADA is selected based on its relevance to the specific decision-making context and is not limited to monetary units alone. For instance, wildfire risk may be quantified by the burned area (hazard) and the exposure could be measured by the population density or the replacement value of homes. Consequently, risk could be expressed in terms of the number of people affected for evacuation planning, or the cost of repairs for property insurance purposes.

Risk has been defined by the International Organization for Standardization as the "effect of uncertainty on objectives" as the potential for consequences when something of value is at stake and the outcome is uncertain, recognizing the diversity of values. Risk can then be quantified as the combination of the probability of a consequence and its magnitude:

$$\text{risk} = \text{probability} \times \text{severity}$$

In the simplest case, $\times$ stands for a multiplication, but more generally, it represents a convolution of the respective distributions of probability and severity. We approximate the $severity$ as follows:

$$\text{severity} = F(\text{hazard intensity}, \text{exposure}, \text{vulnerability}) = \text{exposure} * f_{\text{imp}}(\text{hazard intensity})$$

where f_{imp} is the impact function which parametrizes to what extent an exposure will be affected by a specific hazard. **While the term 'vulnerability function' is broadly used in the modelers community, we adopt the broader term 'impact function'. Impact functions can be vulnerability functions or structural damage functions, but could also be productivity functions or warning levels. This definition also explicitly includes the option of opportunities (i.e. negative damages).**

Using this approach, CLIMADA constitutes a platform to analyse risks of different hazard types in a globally consistent fashion at different resolution levels, at scales from multiple kilometres down to meters, tailored to the specific requirements of the analysis.

References

[IPCC] IPCC: Climate Change 2014: Impacts, Adaptation and Vulnerability. Part A: Global and Sectoral Aspects. Contribution of Working Group II to the Fifth Assessment Report of the Intergovernmental Panel on Climate Change, edited by C. B. Field, V. R. Barros, D. J. Dokken, K. J. Mach, M. D. Mastrandrea, T. E. Bilir, M. Chatterjee, K. L. Ebi, Y. O. Estrada, R. C. Genova, B. Girma, E. S. Kissel, A. N. Levy, S. MacCracken, P. R. Mastrandrea, and L. L. White, Cambridge University Press, United Kingdom and New York, NY, USA., 2014.

1.2.3 Installation

The following sections will guide you through the installation of CLIMADA and its dependencies.

Attention

CLIMADA has a complicated set of dependencies that cannot be installed with `pip` alone. Please follow the installation instructions carefully! We recommend to use a `conda`-based python environment manager such as Mamba or Conda for creating a suitable software environment to execute CLIMADA.

All following instructions should work on any operating system (OS) that is supported by `conda`, including in particular: **Windows**, **macOS**, and **Linux**.

Hint

If you need help with the vocabulary used on this page, refer to the *Glossary*.

Install environment manager

If you haven't already installed an environment management system like Mamba or Conda, you have to do so now. We recommend to use `mamba` (see *Conda as Alternative to Mamba*) which is available in the installer Miniforge (see below).

macOS and Linux

- Open the "Terminal" app, copy-paste the two commands below, and hit enter:

```
curl -L -O "https://github.com/conda-forge/miniforge/releases/latest/download/
↪Miniforge3-$(uname)-$(uname -m).sh"
bash Miniforge3-$(uname)-$(uname -m).sh
```

- Accept the license terms.
- You can confirm the default location.
- Answer 'yes' when asked if if you wish to update your shell profile to automatically initialize conda. **Do not just hit ENTER but first type 'yes'**
- If at some point you encounter `command not found: mamba`, open a new terminal window.
- If you encounter `Run 'mamba init' to be able to run mamba activate/deactivate ...`, please run `mamba init zsh` or `mamba init`.

Windows

- Download the Windows installer at the Install section from Miniforge.
- Execute the installer. This will install Mamba and provide the "Miniforge Prompt" program as a command line replacement.

Python Versions

CLIMADA is primarily tested against a **supported** Python version, but is allowed to run with others. If you follow the installation instructions exactly, you will create an environment with the supported version. Depending on your setup, you are free to choose another allowed version, but we recommend the supported one.

Supported Version	`3.11`
Allowed Versions	`3.10`, `3.11`, `3.12`

Decide on Your Entry Level!

Hint

When mentioning the terms "terminal" or "command line" in the following, we are referring to the "Terminal" apps on macOS or Linux and the "Miniforge Prompt" on Windows.

Depening on your level of expertise, we provide two different approaches:

- If you have never worked with a command line, or if you just want to give CLIMADA a try, follow the *simple instructions*.
- If you want to use the very latest development version of CLIMADA or even develop new CLIMADA code, follow the *advanced instructions*.

Both approaches are not mutually exclusive. After successful installation, you may switch your setup at any time.

Notes on the CLIMADA Petals Package

CLIMADA is divided into two packages, CLIMADA Core (climada_python) and CLIMADA Petals (climada_petals). The Core contains all the modules necessary for probabilistic impact, averted damage, uncertainty and forecast calculations. Data for hazard, exposures and impact functions can be obtained from the *CLIMADA Data API*. Hazard and Exposures subclasses are included as demonstrators only.

> **Attention**
>
> CLIMADA Petals is **not** a standalone module and requires CLIMADA Core to be installed!

CLIMADA Petals contains all the modules for generating data (e.g., `TC_Surge`, `WildFire`, `OpenStreeMap`, ...). New modules are developed and tested here. Some data created with modules from Petals is available to download from the *Data API*. This works with just CLIMADA Core installed. CLIMADA Petals can be used to generate additional data of this type, or to have a look at the tutorials for all data types available from the API.

Both *installation approaches* mentioned above support CLIMADA Petals. If you are unsure whether you need Petals, you can install the Core first and later add Petals in both approaches.

Simple Instructions

These instructions will install the most recent stable version of CLIMADA without cloning its repository.

1. Open the command line. Create a new Conda environment with CLIMADA by executing

   ```
   mamba create -n climada_env -c conda-forge climada
   ```

2. Activate the environment:

   ```
   mamba activate climada_env
   ```

   You should now see `(climada_env)` appear in the beginning of your command prompt. This means the environment is activated.

3. Verify that everything is installed correctly by executing a single test:

   ```
   python -m unittest climada.engine.test.test_impact
   ```

   Executing CLIMADA for the first time will take some time because it will generate a directory tree in your home/user directory. After a while, some text should appear in your terminal. In the end, you should see an "Ok". If so, great! You are good to go.

4. *Optional:* Install CLIMADA Petals into the environment:

   ```
   mamba install -n climada_env -c conda-forge climada-petals
   ```

Advanced Instructions: Installing from source

For advanced Python users or developers of CLIMADA, cloning the CLIMADA repository and installing the package from source.

> **Warning**
>
> If you followed the *Simple Instructions* before, make sure you **either** remove the environment with:

```
mamba env remove -n climada_env
```

before you continue, **or** you use a **different** environment name for the following instructions (e.g. `climada_dev` instead of `climada_env`).

1. If you are using a **Linux** OS, make sure you have `git` installed (Windows and macOS users are good to go once Conda is installed). On Ubuntu and Debian, you may use APT:

   ```
   apt update
   apt install git
   ```

   Both commands will probably require administrator rights, which can be enabled by prepending `sudo`.

2. Create a folder for your code. We will call it **workspace directory**. To make sure that your user can manipulate it without special privileges, use a subdirectory of your user/home directory. Do **not** use a directory that is synchronized by cloud storage systems like OneDrive, iCloud or Polybox!

3. Open the command line and navigate to the workspace directory you created using `cd`. Replace `<path/to/workspace>` with the path of the workspace directory:

   ```
   cd <path/to/workspace>
   ```

4. Clone CLIMADA from its GitHub repository. Enter the directory and check out the branch of your choice. The latest development version will be available under the branch `develop`.

   ```
   git clone https://github.com/CLIMADA-project/climada_python.git
   cd climada_python
   git checkout develop
   ```

5. Create an Conda environment called `climada_env` for installing CLIMADA:

   ```
   mamba create -n climada_env "python=3.11.*"
   ```

   Hint

   Use the wildcard `.*` at the end to allow a downgrade of the bugfix version of Python. This increases compatibility when installing the requirements in the next step.

   Note

   You may choose any of the *allowed Python versions* from the list above.

6. Use the default environment specs in `env_climada.yml` to install all dependencies. Then activate the environment:

   ```
   mamba env update -n climada_env -f requirements/env_climada.yml
   mamba activate climada_env
   ```

7. Install the local CLIMADA source files as Python package using `pip`:

   ```
   python -m pip install -e ./
   ```

> **Hint**
>
> Using a path `./` (referring to the path you are currently located at) will instruct `pip` to install the local files instead of downloading the module from the internet. The `-e` (for "editable") option further instructs `pip` to link to the source files instead of copying them during installation. This means that any changes to the source files will have immediate effects in your environment, and re-installing the module is never required.
>
> Further note that this works only for the source files not for the dependencies. If you change the latter, you will need to update the environment with step 6. !

8. Verify that everything is installed correctly by executing a single test:

```
python -m unittest climada.engine.test.test_impact
```

   Executing CLIMADA for the first time will take some time because it will generate a directory tree in your home/user directory. If this test passes, great! You are good to go.

How to switch branch

Advanced users, or reviewers, may also want to check the feature of a specific branch other than develop. To do so, **assuming you did install CLIMADA in editable mode (`pip install` with the `-e` flag)**, you just have to:

` git fetch git checkout <branch> git pull `

This will work most of the time, except if the target branch defines new dependencies that you don't have already in your environment (as they will not get installed this way), in that case you can install these dependencies yourself, or create a new environment with the *new* requirements from the branch.

If you did not install CLIMADA in editable mode, you can also reinstall CLIMADA from its folder after switching the branch (*pip install [-e] ./*).

Install Developer Dependencies (Optional)

Building the documentation and running the entire test suite of CLIMADA requires additional dependencies which are not installed by default. They are also not needed for using CLIMADA. However, if you want to develop CLIMADA, we strongly recommend you install them.

With the `climada_env` activated, enter the workspace directory and then the CLIMADA repository as above. Then, add the `test` extra specification to the `pip install` command (**mind the quotation marks**,

and see also pip install examples):

```
python -m pip install -e "./[dev]"
```

The CLIMADA Python package defines the following extras:

Extra	Includes Dependencies...
`dev`	for developers, needed for running and evaluating tests and pre-commit actions
`doc`	for building the documentation files from in-source pydoc annotations

The developer dependencies also include pre-commit, which is used to install and run automated, so-called pre-commit hooks before a new commit. In order to use the hooks defined in `.pre-commit-config.yaml`, you need to install the hooks first. With the `climada_env` activated, execute

```
pre-commit install
```

Please refer to the guide on pre-commit hooks for information on how to use this tool.

For executing the pre-defined test scripts in exactly the same way as they are executed by the automated CI pipeline, you will need `make` to be installed. On macOS and on Linux it is pre-installed. On Windows, it can easily be installed with Conda:

```
mamba install -n climada_env make
```

Instructions for running the test scripts can be found in the Testing Guide.

Install CLIMADA Petals (Optional)

If you are unsure whether you need Petals, see the *notes above*.

To install CLIMADA Petals, we assume you have already installed CLIMADA Core with the *advanced instructions* above.

1. Open the command line and navigate to the workspace directory.
2. Clone CLIMADA Petals from its repository. Enter the directory and check out the branch of your choice. The latest development version will be available under the branch `develop`.

   ```
   git clone https://github.com/CLIMADA-project/climada_petals.git
   cd climada_petals
   git checkout develop
   ```

3. Update the Conda environment with the specifications from Petals and activate it:

   ```
   mamba env update -n climada_env -f requirements/env_climada.yml
   mamba activate climada_env
   ```

4. Install the CLIMADA Petals package:

   ```
   python -m pip install -e ./
   ```

Code Editors

JupyterLab

1. Install JupyterLab into the Conda environment:

   ```
   mamba install -n climada_env -c conda-forge jupyterlab
   ```

2. Make sure that the `climada_env` is activated (see above) and then start JupyterLab:

   ```
   mamba activate climada_env
   jupyter-lab
   ```

   JupyterLab will open in a new window of your default browser.

Visual Studio Code (VSCode)

Basic Setup

1. Download and install VSCode following the instructions on https://code.visualstudio.com/.

2. Install the Python and Jupyter extensions. In the left sidebar, select the "Extensions" symbol, enter "Python" in the search bar and click *Install* next to the "Python" extension. Repeat this process for "Jupyter".
3. Open a Jupyter Notebook or create a new one. On the top right, click on *Select Kernel*, select *Python Environments…* and then choose the Python interpreter from the `climada_env`.

See the VSCode docs on Python and Jupyter Notebooks for further information.

> **Hint**
>
> Both of the following setup instructions work analogously for Core and Petals. The specific instructions for Petals are shown in square brackets: []

Workspace Setup

Setting up a workspace for the CLIMADA source code is only available for *advanced installations*.

1. Open a new VSCode window. Below *Start*, click *Open…*, select the `climada_python` [`climada_petals`] repository folder in your workspace directory, and click on *Open* on the bottom right.
2. Click *File* > *Save Workspace As…* and store the workspace settings file next to (**not** in!) the `climada_python` [`climada_petals`] folder. This will enable you to load the workspace and all its specific settings in one go.
3. Open the Command Palette by clicking *View* > *Command Palette* or by using the shortcut keys `Ctrl+Shift+P` (Windows, Linux) / `Cmd+Shift+P` (macOS). Start typing "Python: Select Interpreter" and select it from the dropdown menu. If prompted, choose the option to set the interpreter for the workspace, not just the current folder. Then, choose the Python interpreter from the `climada_env`.

For further information, refer to the VSCode docs on Workspaces.

Test Explorer Setup

After you set up a workspace, you might want to configure the test explorer for easily running the CLIMADA test suite within VSCode.

> **Note**
>
> Please install the additional *test dependencies* before proceeding.

1. In the left sidebar, select the "Testing" symbol, and click on *Configure Python Tests*.
2. Select "pytest" as test framework and then select `climada` [`climada_petals`] as the directory containing the test files.
3. Select "Testing" in the Activity Bar on the left or through *View* > *Testing*. The "Test Explorer" in the left sidebar will display the tree structure of modules, files, test classes and individual tests. You can run individual tests or test subtrees by clicking the Play buttons next to them.
4. By default, the test explorer will show test output for failed tests when you click on them. To view the logs for any test, click on *View* > *Output*, and select "Python Test Log" from the dropdown menu in the view that just opened. If there are errors during test discovery, you can see what's wrong in the "Python" output.

For further information, see the VSCode docs on Python Testing.

Spyder

Installing Spyder into the existing Conda environment for CLIMADA might fail depending on the exact versions of dependencies installed. Therefore, we recommend installing Spyder in a *separate* environment, and then connecting it to a kernel in the original `climada_env`.

1. Follow the Spyder installation instructions. You can follow the "Conda" installation instructions. Keep in mind you are using `mamba`, though!
2. Check the version of the Spyder kernel in the new environment:

   ```
   mamba env export -n spyder-env | grep spyder-kernels
   ```

   This will return a line like this:

   ```
   - spyder-kernels=X.Y.Z=<hash>
   ```

   Copy the part `spyder-kernels=X.Y.Z` (until the second =) and paste it into the following command to install the same kernel version into the `climada_env`:

   ```
   mamba install -n climada_env spyder-kernels=X.Y.Z
   ```

3. Obtain the path to the Python interpreter of your `climada_env`. Execute the following commands:

   ```
   mamba activate climada_env
   python -c "import sys; print(sys.executable)"
   ```

   Copy the resulting path.

4. Open Spyder through the command line:

   ```
   mamba activate spyder-env
   spyder
   ```

5. Set the Python interpreter used by Spyder to the one of `climada_env`. Select *Preferences* > *Python Interpreter* > *Use the following interpreter* and paste the iterpreter path you copied from the `climada_env`.

Apps for working with CLIMADA

To work with CLIMADA, you will need an application that supports Jupyter Notebooks. There are plugins available for nearly every code editor or IDE, but if you are unsure about which to choose, we recommend JupyterLab, Visual Studio Code or Spyder. It is easy to get confused by all the different softwares and their uses so here is an overview of which tools we use for what:

Use	Tools	Description	Useful for
Distribution / manage virtual environment & packages	**Recommended:** Mamba **Alternatives:** Anaconda	• Install climada, manage & use the climada virtual environment, install packages • Anaconda includes Anaconda Navigator, which is a desktop GUI and can be used to launch applications like Jupyter Notebook, Spyder, etc.	Climada Users & Developers
IDE (Integrated Development Environment)	**Recommended:** VSCode **Alternatives:** Spyder, JupyterLab, PyCharm, & many more	• Write and run code • Useful for Developers: - VSCode also has a GUI to commit changes to Git (similar to GitHub Desktop, but in the same place as your code) - VSCode test explorer shows results for individual tests & any classes and files containing those tests (folders display a failure or pass icon)	Climada Users & Developers
Git GUI (Graphical User Interface)	GitHub Desktop, GitKraken	• Provides an interface which keeps track of the branch you're working on, changes you made, etc. • Allows you to commit changes, push to GitHub, etc. without having to use the command line • The code itself is not written using these applications but with your IDE of choice (see above)	Climada Developers
Continuous integration (CI) server	Jenkins	• Automatically checks code changes in GitHub repositories, e.g., when you create a pull request for the develop branch • Performs static code	Climada Developers

FAQs

Answers to frequently asked questions.

Updating CLIMADA

We recommend keeping CLIMADA up-to-date. To update, follow the instructions based on your *installation type*:

- **Simple Instructions:** Update CLIMADA using `mamba`:

```
mamba update -n climada_env -c conda-forge climada
```

- **Advanced Instructions:** Move into your local CLIMADA repository and pull the latest version of your respective branch:

```
cd <path/to/workspace>/climada_python
git pull
```

  Then, update the environment and reinstall the package:

```
mamba env update -n climada_env -f requirements/env_climada.yml
mamba activate climada_env
python -m pip install -e ./
```

  The same instructions apply for CLIMADA Petals.

Installing More Packages

You might use CLIMADA in code that requires more packages than the ones readily available in the CLIMADA Conda environment. If so, **prefer installing these packages via Conda**, and only rely on `pip` if that fails. The default channels of Conda sometimes contain outdated versions. Therefore, use the `conda-forge` channel:

```
mamba install -n climada_env -c conda-forge <package>
```

Only if the desired package (version) is not available, go for `pip`:

```
mamba activate climada_env
python -m pip install <package>
```

Verifying Your Installation

If you followed the installation instructions, you already executed a single unit test. This test, however, will not cover all issues that could occur within your installation setup. If you are unsure if everything works as intended, try running all unit tests. This is only available for *advanced setups*! Move into the CLIMADA repository, activate the environment and then execute the tests:

```
cd <path/to/workspace>/climada_python
mamba activate climada_env
python -m unittest discover -s climada -p "test*.py"
```

Error: `ModuleNotFoundError`

Something is wrong with the environment you are using. After **each** of the following steps, check if the problem is solved, and only continue if it is **not**:

1. Make sure you are working in the CLIMADA environment:

```
mamba activate climada_env
```

2. *Update the Conda environment and CLIMADA*.
3. Conda will notify you if it is not up-to-date. In this case, follow its instructions to update it. Then, repeat the last step and update the environment and CLIMADA (again).
4. Install the missing package manually. Follow the instructions for *installing more packages*.
5. If you reached this point, something is severely broken. The last course of action is to delete your CLIMADA environment:

   ```
   mamba deactivate
   mamba env remove -n climada_env
   ```

   Now repeat the *installation process*.
6. Still no good? Please raise an issue on GitHub to get help.

Logging Configuration

Climada makes use of the standard logging package. By default, the "climada"-`Logger` is detached from `logging.root`, logging to *stdout* with the level set to `WARNING`.

If you prefer another logging configuration, e.g., for using Climada embedded in another application, you can opt out of the default pre-configuration by setting the config value for `logging.climada_style` to `false` in the *configuration file* `climada.conf`.

Changing the logging level can be done in multiple ways:

- Adjust the *configuration file* `climada.conf` by setting a the value of the `global.log_level` property. This only has an effect if the `logging.climada_style` is set to `true` though.
- Set a global logging level in your Python script:

  ```
  import logging
  logging.getLogger('climada').setLevel(logging.ERROR)  # to silence all warnings
  ```

- Set a local logging level in a context manager:

  ```
  from climada.util import log_level
  with log_level(level="INFO"):
      # This also emits all info log messages
      foo()

  # Default logging level again
  bar()
  ```

All three approaches can also be combined.

Conda as Alternative to Mamba

We experienced several issues with the default `conda` package manager lately. This is likely due to the large dependency set of CLIMADA, which makes solving the environment a tedious task. We therefore switched to the more performant `mamba` and recommend using it.

> **Caution**
>
> In theory, you could also use an Anaconda or Miniconda distribution and replace every `mamba` command in this guide with `conda`. In practice, however, `conda` is often unable to solve an environment that `mamba` solves without issues in few seconds.

Error: `operation not permitted`

Conda might report a permission error on macOS Mojave. Carefully follow these instructions: https://github.com/conda/conda/issues/8440#issuecomment-481167572

No `impf_TC` Column in `GeoDataFrame`

This may happen when a demo file from CLIMADA was not updated after the change in the impact function naming pattern from `if_` to `impf_` when CLIMADA v2.2.0 was released. Execute

```
mamba activate climada_env
python -c "import climada; climada.setup_climada_data(reload=True)"
```

The What Now? (Glossary)

You might have become confused about all the names thrown at you. Let's clear that up:

Terminal, Command Line
: A text-only program for interacting with your computer (the old fashioned way). If you are using Miniforge on Windows, the program is called "Miniforge Prompt".

Conda
: A cross-platform package management system. Comes in different varieties (distributions).

Mamba
: The faster reimplementation of the `conda` package manager.

Environment (Programming)
: A setup where only a specific set of modules and programs can interact. This is especially useful if you want to install programs with mutually incompatible requirements.

pip
: The Python package installer.

git
: A popular version control software for programming code (or any text-based set of files).

GitHub
: A website that publicly hosts git repositories.

git Repository
: A collection of files and their entire revision/version history, managed by git.

Cloning
: The process and command (`git clone`) for downloading a git repository.

IDE
: Integrated Development Environment. A fancy source code editor tailored for software development and engineering.

1.2.4 Fast and basic Python introduction

Prepared by G. Aznar Siguan

Most of the examples come from the official Python tutorial: https://docs.python.org/3/tutorial/

Numbers and Strings

The operators +, -, * and / work just like in most other languages:

```
print("Addition: 2 + 2 =", 2 + 2)
print("Substraction: 50 - 5*6 =", 50 - 5 * 6)
print("Use of parenthesis: (50 - 5*6) / 4 =", (50 - 5 * 6) / 4)
print("Classic division returns a float: 17 / 3 =", 17 / 3)
print("Floor division discards the fractional part: 17 // 3 =", 17 // 3)
print("The % operator returns the remainder of the division: 17 % 3 =", 17 % 3)
print("Result * divisor + remainder: 5 * 3 + 2 =", 5 * 3 + 2)
print("5 squared: 5 ** 2 =", 5**2)
print("2 to the power of 7: 2 ** 7 =", 2**7)
```

The integer numbers (e.g. 2, 4, 20) have type int, the ones with a fractional part (e.g. 5.0, 1.6) have type float. Operators with mixed type operands convert the integer operand to floating point:

```
tax = 12.5 / 100
price = 100.50
price * tax
```

Strings can be enclosed in single quotes ('…') or double quotes ("…") with the same result. \ can be used to escape quotes.

If you don't want characters prefaced by \ to be interpreted as special characters, you can use raw strings by adding an r before the first quote.

```
print("spam eggs")  # single quotes
print("doesn't")  # use \' to escape the single quote...
print("doesn't")  # ...or use double quotes instead
print('"Yes," he said.')
print('"Yes," he said.')
print('"Isn\'t," she said.')
```

Strings can be indexed (subscripted), with the first character having index 0.

Indices may also be negative numbers, to start counting from the right. Note that since -0 is the same as 0, negative indices start from -1.

```
word = "Python"
print("word = ", word)
print("Character in position 0: word[0] =", word[0])
print("Character in position 5: word[5] =", word[5])
print("Last character: word[-1] =", word[-1])
print("Second-last character: word[-2] =", word[-2])
print("word[-6] =", word[-6])
```

In addition to indexing, slicing is also supported. While indexing is used to obtain individual characters, slicing allows you to obtain substring:

```
print("Characters from position 0 (included) to 2 (excluded): word[0:2] =", word[0:2])
print("Characters from position 2 (included) to 5 (excluded): word[2:5] =", word[2:5])
```

Lists

Lists can be written as a list of comma-separated values (items) between square brackets. Lists might contain items of different types, but usually the items all have the same type.

Like strings (and all other built-in sequence type), lists can be indexed and sliced:

```
squares = [1, 4, 9, 16, 25]
print("squares: ", squares)
print("Indexing returns the item: squares[0]:", squares[0])
print("squares[-1]:", squares[-1])
print("Slicing returns a new list: squares[-3:]:", squares[-3:])
print("squares[:]:", squares[:])
```

Lists also support operations like concatenation:

```
squares + [36, 49, 64, 81, 100]
```

Unlike strings, which are immutable, lists are a mutable type, i.e. it is possible to change their content:

```
cubes = [1, 8, 27, 65, 125]  # something's wrong here
cubes[3] = 64  # replace the wrong value
cubes.append(216)  # add the cube of 6
cubes.append(7**3)  # and the cube of 7
cubes
```

```
# Note: execution of this cell will fail

# Try to modify a character of a string
word = "Python"
word[0] = "p"
```

List comprehensions provide a concise way to create lists. Common applications are to make new lists where each element is the result of some operations applied to each member of another sequence or iterable, or to create a subsequence of those elements that satisfy a certain condition.

```
squares = []
for x in range(10):
    squares.append(x**2)
squares
```

```
# lambda functions: functions that are not bound to a name, e.g lambda x: x**2
# Map applies a function to all the items in an input_list: map(function_to_apply,␣
↪list_of_inputs)
squares = list(map(lambda x: x**2, range(10)))
squares
```

```
squares = [x**2 for x in range(10)]
squares
```

Tuples

A tuple consists of a number of values separated by commas, for instance:

```
t = 12345, 54321, "hello!"
t[0]
```

```
t
```

```
# Tuples may be nested:
u = t, (1, 2, 3, 4, 5)
u
```

```
# Note: execution of this cell will fail

# Tuples are immutable:
t[0] = 88888
```

```
# but they can contain mutable objects:
v = ([1, 2, 3], [3, 2, 1])
v
```

Tuples are immutable, and usually contain a heterogeneous sequence of elements that are accessed via unpacking or indexing. Lists are mutable, and their elements are usually homogeneous and are accessed by iterating over the list.

```
t = 12345, 54321, "hello!"  # tuple packing
x, y, z = t  # tuple unpacking
x, y, z
```

Sets

A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.

Curly braces or the set() function can be used to create sets.

```
basket = {"apple", "orange", "apple", "pear", "orange", "banana"}
basket  # show that duplicates have been removed
```

```
"orange" in basket  # fast membership testing
```

```
"crabgrass" in basket
```

```
# Demonstrate set operations on unique letters from two words
a = set("abracadabra")
b = set("alacazam")
a  # unique letters in a
```

```
a - b  # letters in a but not in b
```

```
a | b  # letters in a or b or both
```

```
a & b  # letters in both a and b
```

```
a ^ b  # letters in a or b but not both
```

```
# check set documentation
help(set)
```

```
# check which methods can be applied on set
dir(set)
```

```
# Define a new set and try some set methods (freestyle)
```

Dictionaries

Unlike sequences, which are indexed by a range of numbers, dictionaries are indexed by keys, which can be any immutable type; strings and numbers can always be keys.

It is best to think of a dictionary as an unordered set of key: value pairs, with the requirement that the keys are unique (within one dictionary). A pair of braces creates an empty dictionary: {}. Placing a comma-separated list of key:value pairs within the braces adds initial key:value pairs to the dictionary; this is also the way dictionaries are written on output.

```
tel = {"jack": 4098, "sape": 4139}
tel["guido"] = 4127
tel
```

```
tel["jack"]
```

```
del tel["sape"]
```

```
tel["irv"] = 4127
tel
```

```
list(tel.keys())
```

```
sorted(tel.keys())
```

```
"guido" in tel
```

```
"jack" not in tel
```

Functions

We can create a function that writes the Fibonacci series to an arbitrary boundary:

```
def fib(n):  # write Fibonacci series up to n
    """Print a Fibonacci series up to n."""
    a, b = 0, 1  # two assignments in one line
```

(continues on next page)

(continued from previous page)

```
    while a < n:
        print(a, end=" ")
        a, b = b, a + b  # two assignments in one line
    print()
```

```
# Now call the function we just defined:
fib(2000)
```

The value of the function name has a type that is recognized by the interpreter as a user-defined function. This value can be assigned to another name which can then also be used as a function. This serves as a general renaming mechanism:

```
print(fib)
print(type(fib))  # function type
f = fib
f(100)
```

Be careful when using mutable types as inputs in functions, as they might be modified:

```
def dummy(x):
    x += x


xx = 5
print("xx before function call: ", xx)
dummy(xx)
print("xx after function call: ", xx)

yy = [5]
print("yy before function call: ", yy)
dummy(yy)
print("yy after function call: ", yy)
```

Default argument values:

The most useful form is to specify a default value for one or more arguments. This creates a function that can be called with fewer arguments than it is defined to allow. For example:

```
def ask_ok(prompt, retries=4, reminder="Please try again!"):
    while True:
        ok = input(prompt)
        if ok in ("y", "ye", "yes"):
            return True
        if ok in ("n", "no", "nop", "nope"):
            return False
        retries = retries - 1
        if retries < 0:
            raise ValueError("invalid user response")
        print(reminder)
```

```
# This function can be called in several ways:
```

(continues on next page)

(continued from previous page)

```
# giving only the mandatory argument:
ask_ok("Do you really want to quit?")
```

```
# giving one of the optional arguments:
ask_ok("OK to overwrite the file?", 2)
```

```
# or even giving all arguments:
ask_ok("OK to overwrite the file?", 2, "Come on, only yes or no!")
```

Functions can also be called using keyword arguments of the form kwarg=value:

```
ask_ok("OK to overwrite the file?", reminder="Come on, only yes or no!")
```

Default `None` values: `None` default values can be used to handle optional parameters.

```
def test(x=None):
    if x is None:
        print("no x here")
    else:
        print(x)


test()
```

Objects

Example class definition:

```
class Dog:  # same as "class Dog(object)"

    kind = "canine"  # class variable shared by all instances

    def __init__(self, name):  # initialization method
        self.name = name  # instance variable unique to each instance
        self.tricks = []  # creates a new empty list for each dog

    def add_trick(self, trick):  # class method
        self.tricks.append(trick)
```

When a class defines an *_init_*() method, class instantiation automatically invokes *_init_*() for the newly-created class instance:

```
d = Dog(
    "Fido"
)  # creates a new instance of the class and assigns this object to the local␣
↪variable d
d.name
```

```
e = Dog(
    "Buddy"
)  # creates a new instance of the class and assigns this object to the local␣
↪variable e
```

(continues on next page)

(continued from previous page)

```
d.add_trick("roll over")
e.add_trick("play dead")
```

```
d.tricks  # unique to d
```

```
e.tricks  # unique to e
```

```
d.kind  # shared by all dogs
```

```
e.kind  # shared by all dogs
```

Inheritance:

The syntax for a derived class definition looks like this:

```
class DerivedClassName(BaseClassName)
```

A derived class can override any methods of its base class or classes, and a method can call the method of a base class with the same name. Example:

```
class Animal:  # base class

    def __init__(self, kind):
        self.kind = kind
        self.tricks = []

    def add_trick(self, trick):  # class method
        self.tricks.append(trick)


class Dog(Animal):  # derived class

    def __init__(self):  # override of __init__ base method
        super(Dog, self).__init__(
            "canine"
        )  # call Animal __init__ method with input string
```

```
fido = Dog()  # fido is automatically an animal of kind 'canine'
print(fido.kind)
fido.add_trick("play dead")  # Dog class can use Animal class
print(fido.tricks)
```

Python supports a form of multiple inheritance as well. A class definition with multiple base classes looks like this:

```
class DerivedClassName(Base1, Base2, Base3):
```

Private Variables and Methods:

"Private" instance variables that cannot be accessed except from inside an object don't exist in Python. However, there is a convention that is followed by most Python code: a name prefixed with an underscore (e.g. _spam) should be treated as a non-public part of the API (whether it is a function, a method or a data member).

Example of internal class use of private method __update. The user is not meant to use __update, but update. However, *_update can be used internally to be called from the* _init method:

```
class Mapping:
    def __init__(self, iterable):
        self.items_list = []
        self.__update(iterable)

    def update(self, iterable):
        for item in iterable:
            self.items_list.append(item)

    __update = update  # private copy of original update() method


class MappingSubclass(Mapping):

    def update(self, keys, values):
        # provides new signature for update()
        # but does not break __init__()
        for item in zip(keys, values):
            self.items_list.append(item)
```

CHAPTER

TWO

USER GUIDE

This user guide contains all the detailed tutorials about the different parts of CLIMADA. If you are a new user, we advise you to have a look at the 10 minutes CLIMADA which introduces the basics briefly, or the full Overview which goes more in depth.

You can then go on to more specific tutorial about Hazard, Exposures or Impact or advanced usage such as Uncertainty Quantification

2.1 10 minutes CLIMADA

This is a brief introduction to CLIMADA that showcases CLIMADA's key building block, the impact calculation. For more details and features of the impact calculation, please check out the more detailed *CLIMADA Overview*.

2.1.1 Key ingredients in a CLIMADA impact calculation

For CLIMADA's impact calculation, we have to specify the following ingredients:

- **Hazard**: The hazard object entails event-based and spatially-resolved information of the intensity of a natural hazard. It contains a probabilistic event set, meaning that is a set of several events, each of which is associated to a frequency corresponding to the estimated probability of the occurence of the event.
- **Exposure**: The exposure information provides the location and the number and/or value of objects (e.g., humans, buildings, ecosystems) that are exposed to the hazard.
- **Vulnerability**: The impact or vunerability function models the average impact that is expected for a given exposure value and given hazard intensity.

2.1.2 Exemplary impact calculation

We exemplify the impact calculation and its key ingredients with an analysis of the risk of tropical cyclones on several assets in Florida.

Hazard objects

First, we read a demo hazard file that includes information about several tropical cyclone events.

```
from climada.hazard import Hazard
from climada.util import HAZ_DEMO_H5

haz = Hazard.from_hdf5(HAZ_DEMO_H5)

# to hide the warnings
import warnings
```

(continues on next page)

(continued from previous page)

```
warnings.filterwarnings("ignore")
```

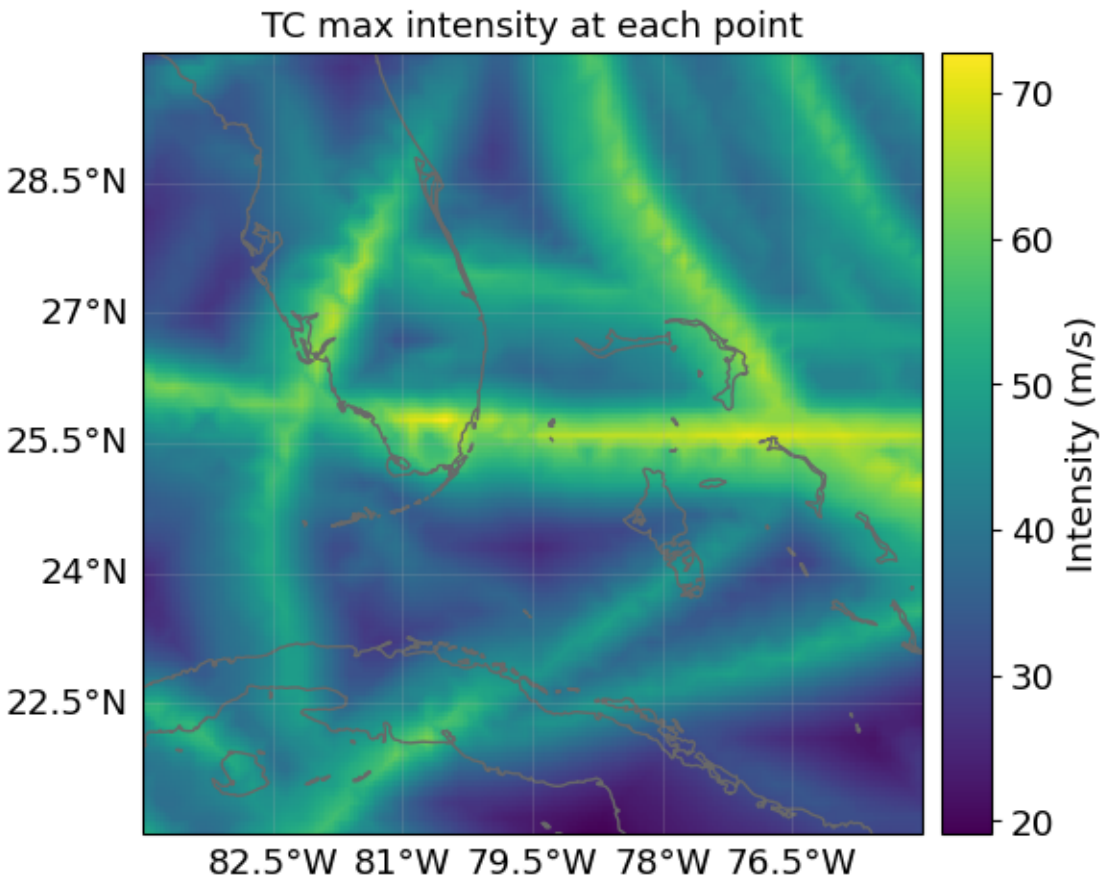
We can infer some information from the Hazard object. The central piece of the hazard object is a sparse matrix at `haz.intensity` that contains the hazard intensity values for each event (axis 0) and each location (axis 1).

```
print(
    f"The hazard object contains {haz.intensity.shape[0]} events. \n"
    f"The maximal intensity contained in the Hazard object is {haz.intensity.max():.
↪2f} {haz.units}. \n"
    f"The first event was observed in a time series of {int(1/haz.frequency[0])} {haz.
↪frequency_unit[2:]}s, \n"
    f"which is why CLIMADA estimates an annual probability of {haz.frequency[0]:.4f}␣
↪for the occurence of this event."
)
```

```
The hazard object contains 216 events.
The maximal intensity contained in the Hazard object is 72.75 m/s.
The first event was observed in a time series of 185 years,
which is why CLIMADA estimates an annual probability of 0.0054 for the occurence of␣
↪this event.
```

The probabilistic event set and its single events can be plotted. For instance, below we plot maximal intensity per grid point over the whole event set.

```
haz.plot_intensity(0, figsize=(6, 6));
```



Exposure objects

Now, we read a demo expopure file containing the location and value of a number of exposed assets in Florida.

```
from climada.entity import Exposures
from climada.util.constants import EXP_DEMO_H5

exp = Exposures.from_hdf5(EXP_DEMO_H5)
```

```
2025-01-21 15:38:13,269 - climada.entity.exposures.base - INFO - Reading /Users/
↪vgebhart/climada/demo/data/exp_demo_today.h5
```

We can print some basic information about the exposure object. The central information of the exposure object is contained in a geopandas.GeoDataFrame at `exp.gdf`.

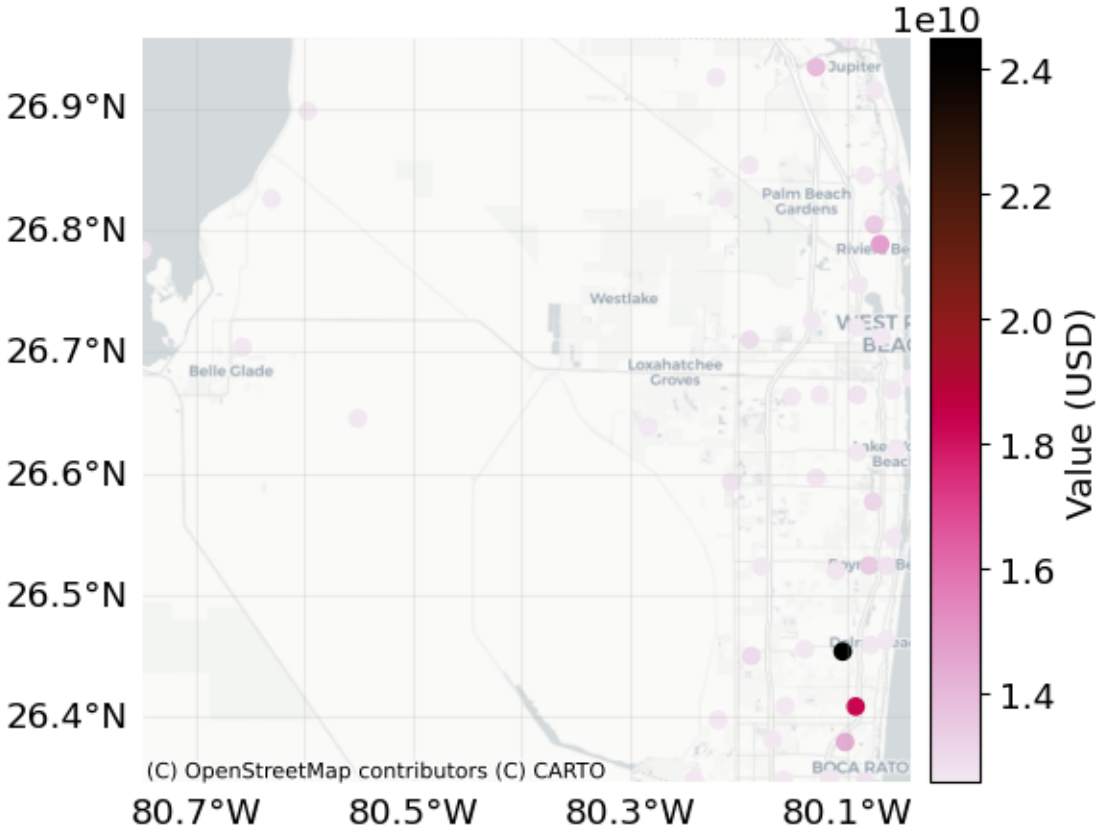
```
print(
    f"In the exposure object, a total amount of {exp.value_unit} {exp.gdf.value.sum()␣
↪/ 1_000_000_000:.2f}B"
    f" is distributed among {exp.gdf.shape[0]} points."
)
```

```
In the exposure object, a total amount of USD 657.05B is distributed among 50 points.
```

We can plot the different exposure points on a map.

```
exp.plot_basemap(figsize=(6, 6));
```

```
2025-01-21 15:39:38,249 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.
2025-01-21 15:39:38,498 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.
```

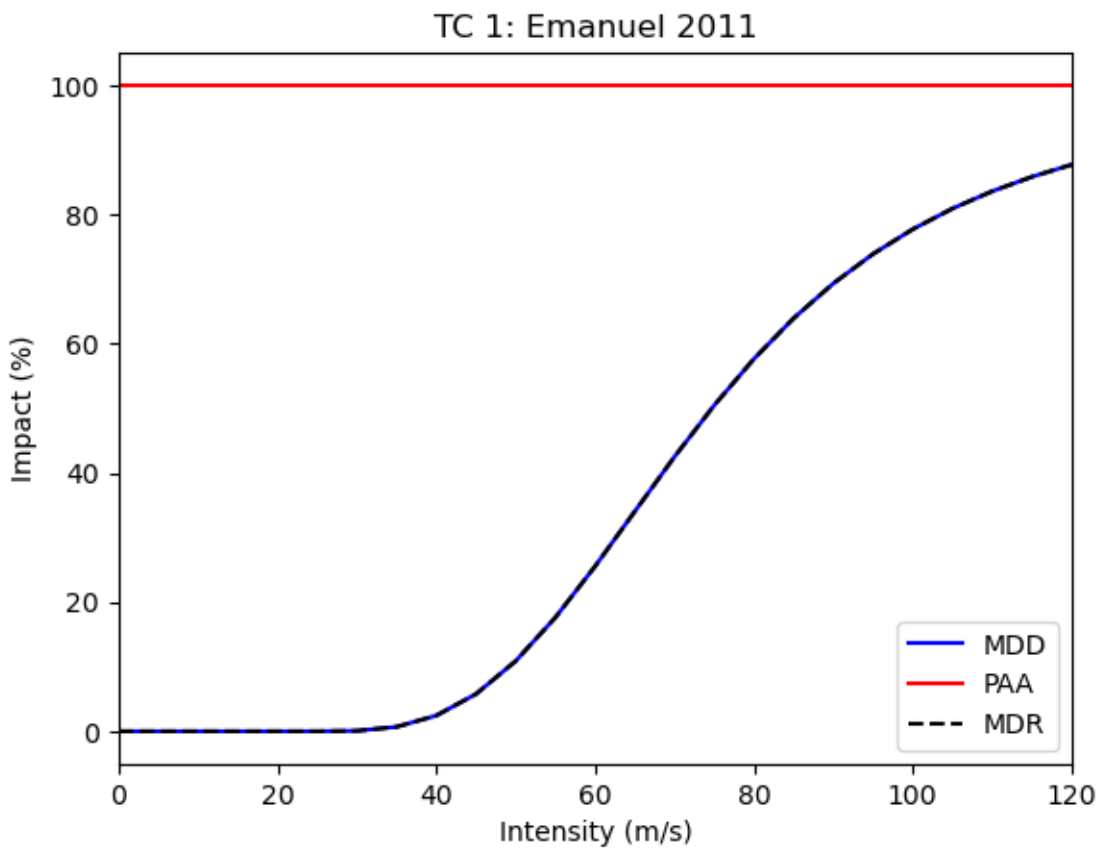


Impact Functions

To model the impact to the exposure that is caused by the hazard, CLIMADA makes use of an impact function. This function relates both percentage of assets affected (PAA, red line below) and the mean damage degree (MDD, blue line below), to the hazard intensity. The multiplication of PAA and MDD result in the mean damage ratio (MDR, black dashed line below), that relates the hazard intensity to corresponding relative impact values. Finally, a multiplication with the exposure values results in the total impact.

Below, we read and plot a standard impact function for tropical cyclones.

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone

impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_set = ImpactFuncSet([impf_tc])
impf_set.plot();
```



Impact calculation

Having defined hazard, exposure, and impact function, we can finally perform the impact calcuation.

```
from climada.engine import ImpactCalc

imp = ImpactCalc(exp, impf_set, haz).impact(save_mat=True)
```

```
2025-01-21 15:43:22,682 - climada.entity.exposures.base - INFO - Matching 50␣
↪exposures with 2500 centroids.
2025-01-21 15:43:22,683 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2025-01-21 15:43:22,686 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪250 assets (>0) and 216 events.
2025-01-21 15:43:22,687 - climada.engine.impact_calc - INFO - cover and/or deductible␣
↪columns detected, going to calculate insured impact
```

The Impact object contains the results of the impact calculation (including event- and location-wise impact information when `save_mat=True`).

```
print(
    f"The total expected annual impact over all exposure points is {imp.unit} {imp.
↪aai_agg / 1_000_000:.2f} M. \n"
    f"The largest estimated single-event impact is {imp.unit} {max(imp.at_event) / 1_
↪000_000_000:.2f} B. \n"
```

(continues on next page)

(continued from previous page)

```
    f"The largest expected annual impact for a single location is {imp.unit} {max(imp.
↪eai_exp) / 1_000_000:.2f} M. \n"
)
```

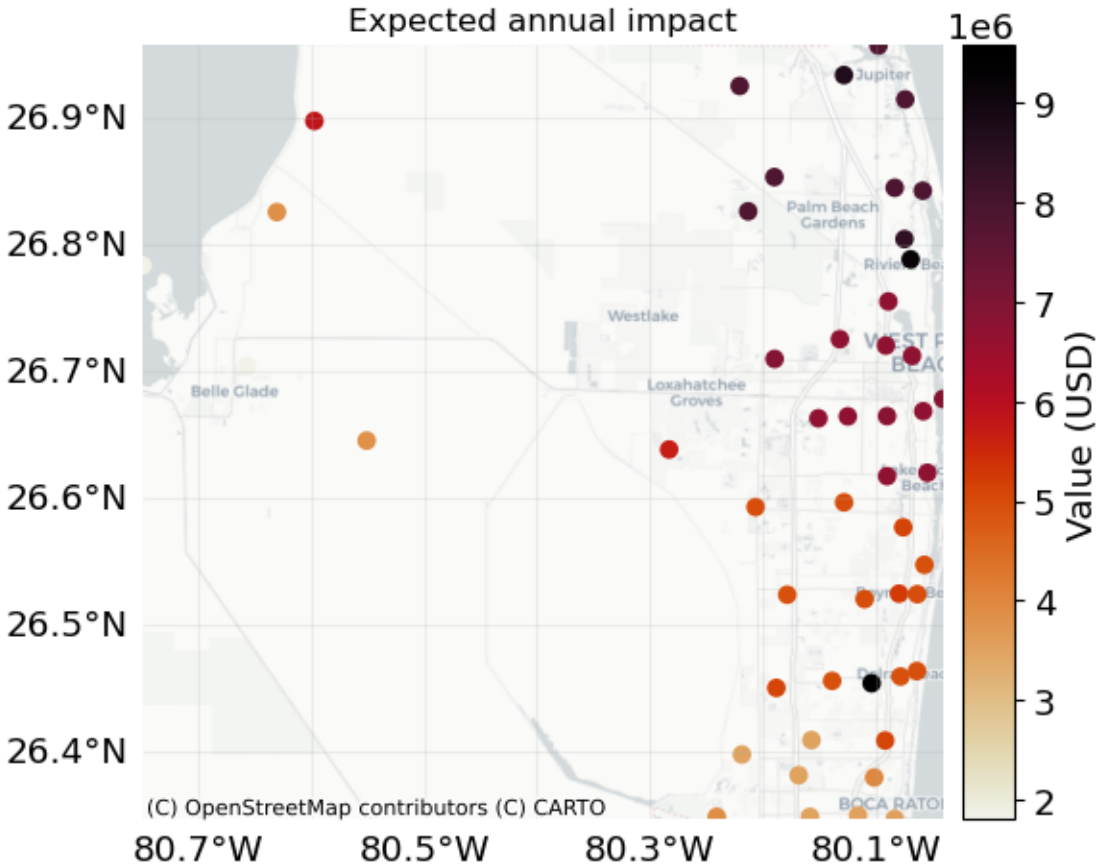
```
The total expected annual impact over all exposure points is USD 288.90 M.
The largest estimated single-event impact is USD 20.96 B.
The largest expected annual impact for a single location is USD 9.58 M.
```

Several visualizations of impact objects are available. For instance, we can plot the expected annual impact per location on a map.

```
imp.plot_basemap_eai_exposure(figsize=(6, 6))
```

```
2025-01-21 15:44:16,514 - climada.util.coordinates - INFO - Setting geometry points.
2025-01-21 15:44:16,518 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2025-01-21 15:44:16,771 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
<GeoAxes: title={'center': 'Expected annual impact'}>
```



2.1.3 Further CLIMADA features

CLIMADA offers several additional features and modules that complement its basic impact and risk calculation, among which are

- uncertainty and sensitivity analysis
- adaptation option appraisal and cost benefit analysis
- several tools for providing hazard objects such as tropical cyclones, floods, or winter storms; and exposure objects such as Litpop, or open street maps
- impact function calibration methods

We end this introduction with a simple adaptation measure analysis.

Adaptation measure analysis

Consider a simple adaptation measure that results in a 10% decrease in the percentage of affected assets (PAA) decreases and a 20% decrease in the mean damage degree (MDD). We apply this measure and recompute the impact.

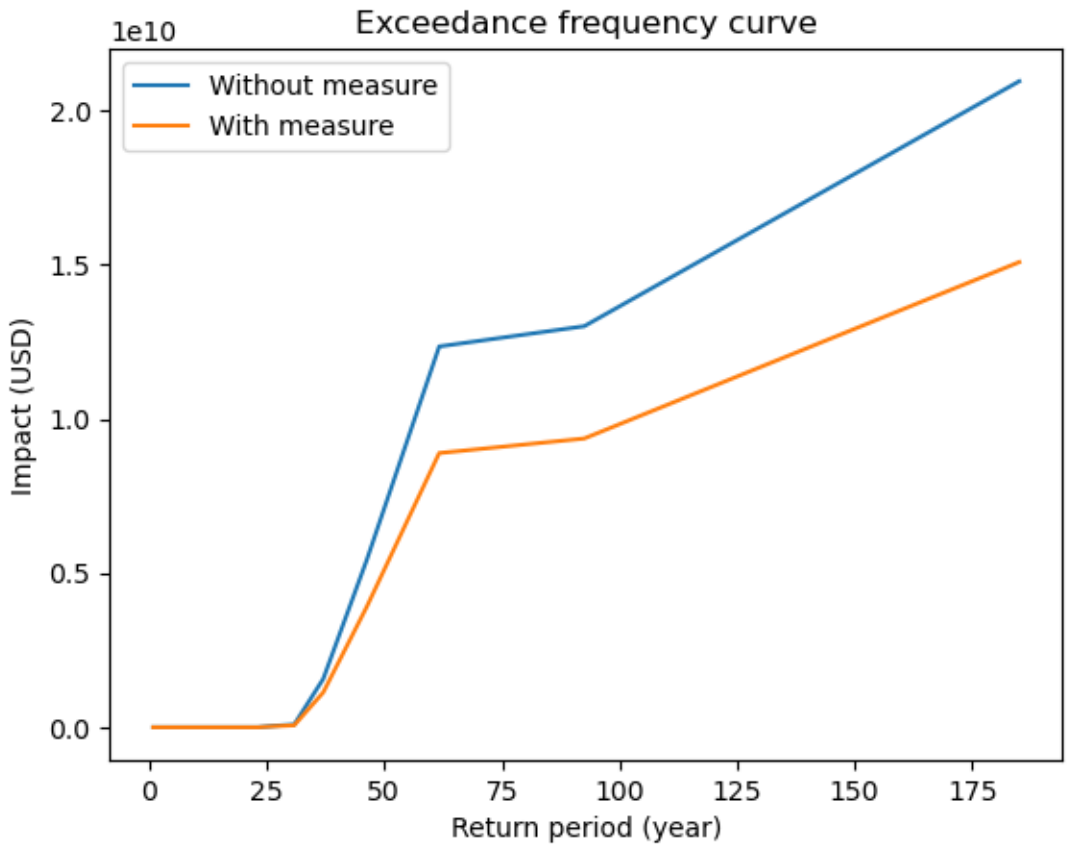
```
from climada.entity.measures import Measure

meas = Measure(haz_type="TC", paa_impact=(0.9, 0), mdd_impact=(0.8, 0))

new_exp, new_impfs, new_haz = meas.apply(exp, impf_set, haz)
new_imp = ImpactCalc(new_exp, new_impfs, new_haz).impact()
```

```
2025-01-21 15:49:48,642 - climada.entity.exposures.base - INFO - Exposures matching␣
↪centroids already found for TC
2025-01-21 15:49:48,643 - climada.entity.exposures.base - INFO - Existing centroids␣
↪will be overwritten for TC
2025-01-21 15:49:48,643 - climada.entity.exposures.base - INFO - Matching 50␣
↪exposures with 2500 centroids.
2025-01-21 15:49:48,645 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2025-01-21 15:49:48,648 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪250 assets (>0) and 216 events.
2025-01-21 15:49:48,648 - climada.engine.impact_calc - INFO - cover and/or deductible␣
↪columns detected, going to calculate insured impact
```

To analyze the effect of the adaptation measure, we can, for instance, plot the impact exceedance frequency curves that describe, according to the given data, how frequent different impacts thresholds are expected to be exceeded.

```
ax = imp.calc_freq_curve().plot(label="Without measure")
new_imp.calc_freq_curve().plot(axis=ax, label="With measure")
ax.legend()
```

```
<matplotlib.legend.Legend at 0x17f5ef710>
```



2.2 CLIMADA overview

2.2.1 Introduction

What is CLIMADA?

CLIMADA is a fully probabilistic climate risk assessment tool. It provides a framework for users to combine exposure, hazard and vulnerability or impact data to calculate risk. Users can create probabilistic impact data from event sets, look at how climate change affects these impacts, and see how effectively adaptation measures can change them. CLIMADA also allows for studies of individual events, historical event sets and forecasts.

The model is highly customisable, meaning that users can work with out-of-the-box data provided for different hazards, population and economic exposure, or can provide their own data for part or all of the analysis. The pre-packaged data make CLIMADA particularly useful for users who focus on just one element of risk, since CLIMADA can 'fill in the gaps' for hazard, exposure or vulnerability in the rest of the analysis.

The model core is designed to give as much flexibility as possible when describing the elements of risk, meaning that CLIMADA isn't limited to particular hazards, exposure types or impacts. We love to see the model applied to new problems and contexts.

CLIMADA provides classes, methods and data for exposure, hazard and impact functions (also called vulnerability functions), plus a financial model and a framework to analyse adaptation measures. Additional classes and data for common uses, such as economic exposures or tropical storms and tutorials for every class are available: see the *CLIMADA features* section below.

This tutorial

This tutorial is for people new to CLIMADA who want to get a high level understanding of the model and work through an example risk analysis. It will list the current features of the model, and go through a complete CLIMADA analysis to give an idea of how the model works. Other tutorials go into more detail about different model components and individual hazards.

Resources beyond this tutorial

- *Installation guide* - go here if you've not installed the model yet
- CLIMADA Read the Docs home page - for all other documentation
- *List of CLIMADA's features and associated tutorials*
- CLIMADA GitHub develop branch documentation for the very latest versions of code and documentation
- CLIMADA paper GitHub repository - for publications using CLIMADA

2.2.2 CLIMADA features

A risk analysis with CLIMADA can include

1. the statistical risk to your exposure from a set of events,
2. how it changes under climate change, and
3. a cost-benefit analysis of adaptation measures.

CLIMADA is flexible: the "statistical risk" above could be describing the annual expected insured flood losses to a property portfolio, the number of people displaced by an ensemble of typhoon forecasts, the annual disruption to a railway network from landslides, or changes to crop yields.

Users from risk-analysis backgrounds will be familiar with describing the impact of events by combining exposure, hazard and an impact function (or vulnerability curve) that combines the two to describe a hazard's effects. A CLIMADA analysis uses the same approach but wraps the exposures and their impact functions into a single `Entity` class, along with discount rates and adaptation options (see the below tutorials for more on CLIMADA's financial model).

CLIMADA's `Impact` object is used to analyse events and event sets, whether this is the impact of a single wildfire, or the global economic risk from tropical cyclones in 2100.

CLIMADA is divided into two parts (two repositories):

1. the core climada_python contains all the modules necessary for the probabilistic impact, the averted damage, uncertainty and forecast calculations. Data for hazard, exposures and impact functions can be obtained from the *data API*. *Litpop* is included as demo Exposures module, and *Tropical cyclones* is included as a demo Hazard module.
2. the petals climada_petals contains all the modules for generating data (e.g., TC_Surge, WildFire, OpenStreeMap, …). Most development is done here. The petals builds-upon the core and does not work as a stand-alone.

CLIMADA classes

This is a full directory of tutorials for CLIMADA's classes to use as a reference. You don't need to read all this to do this tutorial, but it may be useful to refer back to.

Core (climada_python):

- ***Hazard***: a class that stores sets of geographic hazard footprints, (e.g. for wind speed, water depth and fraction, drought index), and metadata including event frequency. Several predefined extensions to create particular hazards from particular datasets and models are included with CLIMADA:

  - *Tropical cyclone wind*: global hazard sets for tropical cyclone events, constructing statistical wind fields from storm tracks. Subclasses include methods and data to calculate historical wind footprints, create forecast enembles from ECMWF tracks, and create climatological event sets for different climate scenarios.
  - *European windstorms*: includes methods to read and plot footprints from the Copernicus WISC dataset and for DWD and ICON forecasts.

- ***Entity***: this is a container that groups CLIMADA's socio-economic models. It's is where the Exposures and Impact Functions are stored, which can then be combined with a hazard for a risk analysis (using the Engine's Impact class). It is also where Discount Rates and Measure Sets are stored, which are used in adaptation cost-benefit analyses (using the Engine's CostBenefit class):
  - *Exposures*: geolocated exposures. Each exposure is associated with a value (which can be a dollar value, population, crop yield, etc), information to associate it with impact functions for the relevant hazard(s) (in the Entity's ImpactFuncSet), a geometry, and other optional properties such as deductables and cover. Exposures can be loaded from a file, specified by the user, or created from regional economic models accessible within CLIMADA, for example:
    * *LitPop*: regional economic model using nightlight and population maps together with several economic indicators
    * *Polygons_lines*: use CLIMADA Impf you have your exposure in the form of shapes/polygons or in the form of lines.
  - *ImpactFuncSet*: functions to describe the impacts that hazards have on exposures, expressed in terms of e.g. the % dollar value of a building lost as a function of water depth, or the mortality rate for over-70s as a function of temperature. CLIMADA provides some common impact functions, or they can be user-specified. The following is an incomplete list:
    * ImpactFunc: a basic adjustable impact function, specified by the user
    * IFTropCyclone: impact functions for tropical cyclone winds
    * IFRiverFlood: impact functions for river floods
    * IFStormEurope: impact functions for European windstorms
  - *DiscRates*: discount rates per year
  - *MeasureSet*: a collection of Measure objects that together describe any adaptation measures being modelled. Adaptation measures are described by their cost, and how they modify exposure, hazard, and impact functions (and have have a method to do these things). Measures also include risk transfer options.

- ***Engine***: the CLIMADA Engine contains the Impact and CostBenefit classes, which are where the main model calculations are done, combining Hazard and Entity objects.
  - *Impact*: a class that stores CLIMADA's modelled impacts and the methods to calculate them from Exposure, Impact Function and Hazard classes. The calculations include average annual impact, expected annual impact by exposure item, total impact by event, and (optionally) the impact of each event on each exposure point. Includes statistical and plotting routines for common analysis products.
  - *Impact_data*: The core functionality of the module is to read disaster impact data as downloaded from the International Disaster Database EM-DAT (www.emdat.be) and produce a CLIMADA Impact()-instance from it. The purpose is to make impact data easily available for comparison with simulated impact inside CLIMADA, e.g. for calibration purposes.
  - *CostBenefit*: a class to appraise adaptation options. It uses an Entity's MeasureSet to calculate new Impacts based on their adjustments to hazard, exposure, and impact functions, and returns statistics and plotting routines to express cost-benefit comparisons.
  - *Unsequa*: a module for uncertainty and sensitivity analysis.

  - *Unsequa_helper*: The InputVar class provides a few helper methods to generate generic uncertainty input variables for exposures, impact function sets, hazards, and entities (including measures cost and disc rates). This tutorial complements the general tutorial on the uncertainty and sensitivity analysis module unsequa.
  - *Forecast*: This class deals with weather forecasts and uses CLIMADA ImpactCalc.impact() to forecast impacts of weather events on society. It mainly does one thing: It contains all plotting and other functionality that are specific for weather forecasts, impact forecasts and warnings.

climada_petals:

- ***Hazard***:
  - Storm surge: Tropical cyclone surge from linear wind-surge relationship and a bathtub model.
  - River flooding: global water depth hazard for flood, including methods to work with ISIMIP simulations.
  - Crop modelling: combines ISIMIP crop simulations and UN Food and Agrigultre Organization data. The module uses crop production as exposure, with hydrometeorological 'hazard' increasing or decreasing production.
  - Wildfire (global): This class is used to model the wildfire hazard using the historical data available and creating synthetic fires which are summarized into event years to establish a comprehensiv probabilistic risk assessment.
  - Landslide: This class is able to handle two different types of landslide source files (in one case, already the finished product of some model output, in the other case just a historic data collection).
  - TCForecast: This class extends the TCTracks class with methods to download operational ECMWF ensemble tropical storm track forecasts, read the BUFR files they're contained in and produce a TCTracks object that can be used to generate TropCyclone hazard footprints.
  - Emulator:Given a database of hazard events, this module climada.hazard.emulator provides tools to subsample events (or time series of events) from that event database.
  - Drought (global): tutorial under development
- ***Entity***:
  - *Exposures*:
    * BlackMarble: regional economic model from nightlight intensities and economic indicators (GDP, income group). Largely succeeded by LitPop.
    * OpenStreetMap: CLIMADA provides some ways to make use of the entire OpenStreetMap data world and to use those data within the risk modelling chain of CLIMADA as exposures.
- ***Engine***:
  - SupplyChain: This class allows assessing indirect impacts via Input-Ouput modeling.

This list will be updated periodically along with new CLIMADA releases. To see the latest, development version of all tutorials, see the tutorials page on the CLIMADA GitHub.

2.2.3 Tutorial: an example risk assessment

This example will work through a risk assessment for tropical storm wind in Puerto Rico, constructing hazard, exposure and vulnerability and combining them to create an Impact object. Everything you need for this is included in the main CLIMADA installation and additional data will be downloaded by the scripts as required.

2.2.4 Hazard

Hazards are characterized by their frequency of occurrence and the geographical distribution of their intensity. The `Hazard` class collects events of the same hazard type (e.g. tropical cyclone, flood, drought, …) with intensity values over the same geographic centroids. They might be historical events or synthetic.

See the *Hazard tutorial* to learn about the Hazard class in more detail, and the *CLIMADA features* section of this document to explore tutorials for different hazards, including *tropical cyclones*, as used here.

Tropical cyclones in CLIMADA and the `TropCyclone` class work like any hazard, storing each event's wind speeds at the geographic centroids specified for the class. Pre-calculated hazards can be loaded from files (see the *full Hazard tutorial*, but they can also be modelled from a storm track using the `TCTracks` class, based on a storm's parameters at each time step. This is how we'll construct the hazards for our example.

So before we create the hazard, we will create our storm tracks and define the geographic centroids for the locations we want to calculate hazard at.

Storm tracks

Storm tracks are created and stored in a separate class, `TCTracks`. We use its method `from_ibtracs_netcdf` to create the tracks from the IBTRaCS storm tracks archive. In the next block we will download the full dataset, which might take a little time. However, to plot the whole dataset takes too long (see the second block), so we choose a shorter time range here to show the function. See the *full TropCyclone tutorial* for more detail and troubleshooting.

```
import numpy as np
from climada.hazard import TCTracks
import warnings  # To hide the warnings

warnings.filterwarnings("ignore")

tracks = TCTracks.from_ibtracs_netcdf(
    provider="usa", basin="NA"
)  # Here we download the full dataset for the analysis
# afterwards (e.g. return period), but you can also use "year_range" to adjust the␣
↪range of the dataset to be downloaded.
# While doing that, you need to make sure that the year 2017 is included if you want␣
↪to run the blocks with the codes
# subsetting a specific tropic cyclone, which happened in 2017. (Of course, you can␣
↪also change the subsetting codes.)
```

```
2022-03-21 14:31:20,322 - climada.hazard.tc_tracks - WARNING - 1122 storm events are␣
↪discarded because no valid wind/pressure values have been found: 1851175N26270,␣
↪1851181N19275, 1851187N22262, 1851192N12300, 1851214N14321, ...
2022-03-21 14:31:20,345 - climada.hazard.tc_tracks - WARNING - 139 storm events are␣
↪discarded because only one valid timestep has been found: 1852232N21293,␣
↪1853242N12336, 1855236N12304, 1856221N25277, 1856235N13302, ...
2022-03-21 14:31:22,766 - climada.hazard.tc_tracks - INFO - Progress: 10%
2022-03-21 14:31:25,059 - climada.hazard.tc_tracks - INFO - Progress: 20%
2022-03-21 14:31:27,491 - climada.hazard.tc_tracks - INFO - Progress: 30%
2022-03-21 14:31:30,067 - climada.hazard.tc_tracks - INFO - Progress: 40%
2022-03-21 14:31:32,415 - climada.hazard.tc_tracks - INFO - Progress: 50%
2022-03-21 14:31:34,829 - climada.hazard.tc_tracks - INFO - Progress: 60%
2022-03-21 14:31:37,482 - climada.hazard.tc_tracks - INFO - Progress: 70%
2022-03-21 14:31:39,976 - climada.hazard.tc_tracks - INFO - Progress: 80%
2022-03-21 14:31:42,307 - climada.hazard.tc_tracks - INFO - Progress: 90%
```

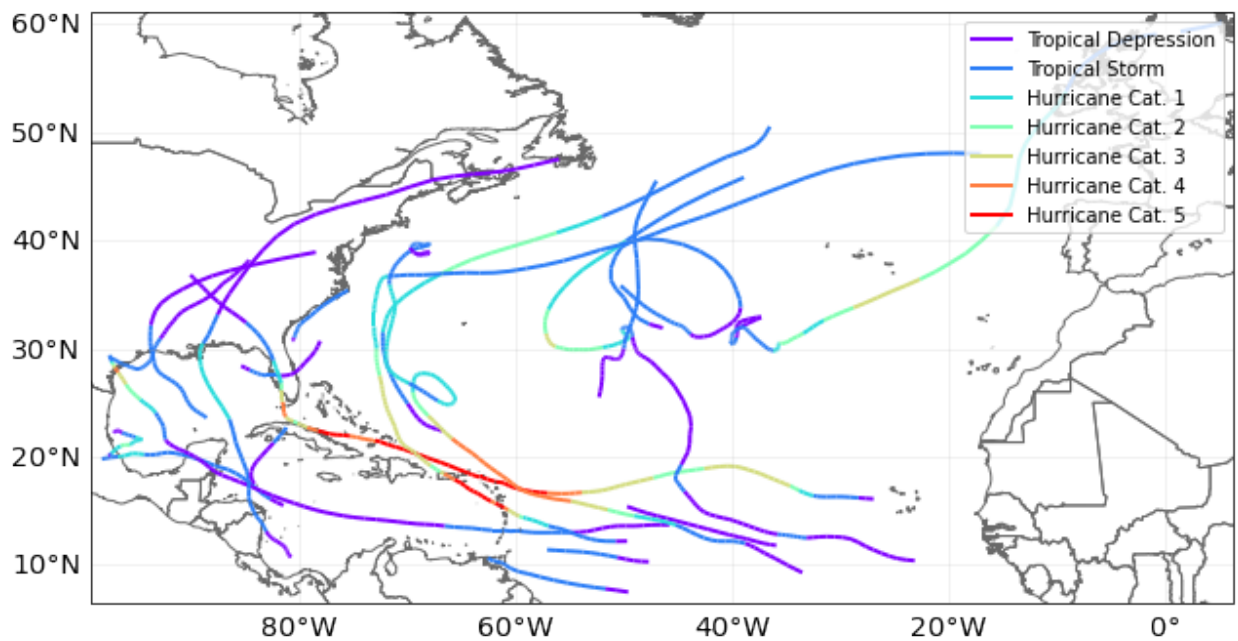
(continues on next page)

(continued from previous page)

```
2022-03-21 14:31:44,580 - climada.hazard.tc_tracks - INFO - Progress: 100%
2022-03-21 14:31:45,780 - climada.hazard.tc_tracks - INFO - Progress: 10%
2022-03-21 14:31:45,833 - climada.hazard.tc_tracks - INFO - Progress: 21%
2022-03-21 14:31:45,886 - climada.hazard.tc_tracks - INFO - Progress: 31%
2022-03-21 14:31:45,939 - climada.hazard.tc_tracks - INFO - Progress: 42%
2022-03-21 14:31:45,992 - climada.hazard.tc_tracks - INFO - Progress: 52%
2022-03-21 14:31:46,048 - climada.hazard.tc_tracks - INFO - Progress: 63%
2022-03-21 14:31:46,100 - climada.hazard.tc_tracks - INFO - Progress: 73%
2022-03-21 14:31:46,150 - climada.hazard.tc_tracks - INFO - Progress: 84%
2022-03-21 14:31:46,203 - climada.hazard.tc_tracks - INFO - Progress: 94%
2022-03-21 14:31:46,232 - climada.hazard.tc_tracks - INFO - Progress: 100%
```

This will load all historical tracks in the North Atlantic into the `tracks` object (since we set `basin='NA'`). The `TCTracks.plot` method will plot the downloaded tracks, though there are too many for the plot to be very useful:

```
# plotting tracks can be very time consuming, depending on the number of tracks. So␣
↪we choose only a few here, by limiting the time range to one year
tracks_2017 = TCTracks.from_ibtracs_netcdf(
    provider="usa", basin="NA", year_range=(2017, 2017)
)
tracks_2017.plot();  # This may take a very long time
```



It's also worth adding additional time steps to the tracks (though this can be memory intensive!). Most tracks are reported at 3-hourly intervals (plus a frame at landfall). Event footprints are calculated as the maximum wind from any time step. For a fast-moving storm these combined three-hourly footprints give quite a rough event footprint, and it's worth adding extra frames to smooth the footprint artificially (try running this notebook with and without this interpolation to see the effect):

```
tracks.equal_timestep(time_step_h=0.5)
```

```
2022-03-21 14:32:39,466 - climada.hazard.tc_tracks - INFO - Interpolating 1049 tracks␣
↪to 0.5h time steps.
```

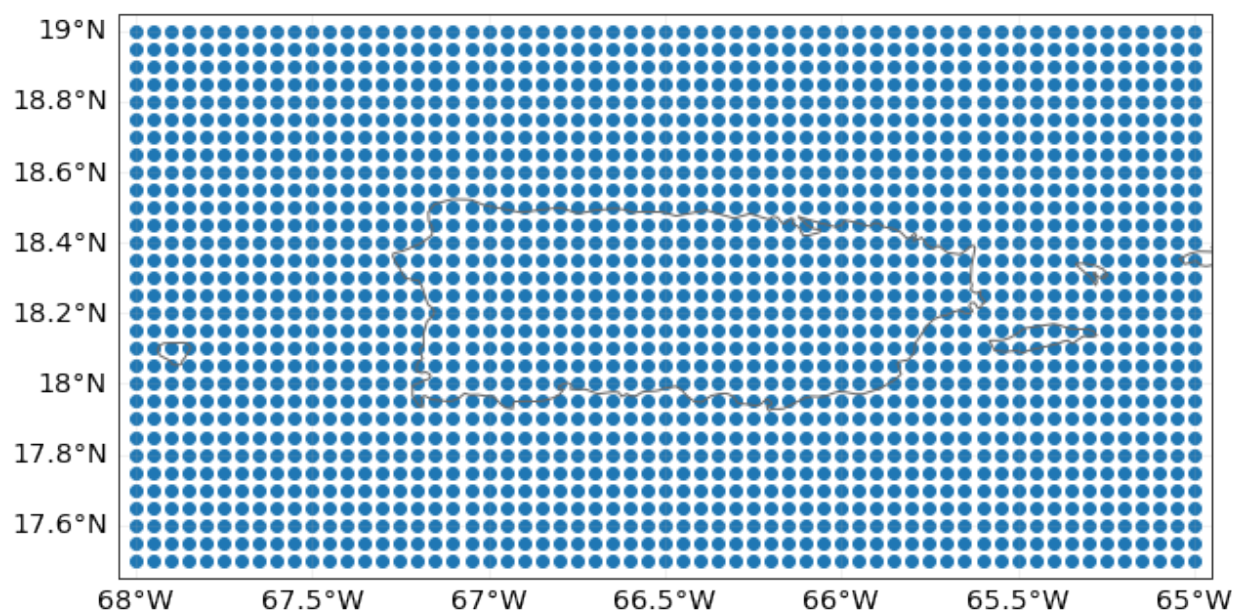
Now, irresponsibly for a risk analysis, we're only going to use these historical events: they're enough to demonstrate CLIMADA in action. A proper risk analysis would expand it to include enough events for a statistically robust climatology. See the *full TropCyclone tutorial* for CLIMADA's stochastic event generation.

Centroids

A hazard's centroids can be any set of locations where we want the hazard to be evaluated. This could be the same as the locations of your exposure, though commonly it is on a regular lat-lon grid (with hazard being imputed to exposure between grid points).

Here we'll set the centroids as a 0.1 degree grid covering Puerto Rico. Centroids are defined by a `Centroids` class, which has the `from_pnt_bounds` method for generating regular grids and a `plot` method to inspect the centroids.

```
from climada.hazard import Centroids

min_lat, max_lat, min_lon, max_lon = 17.5, 19.0, -68.0, -65.0
cent = Centroids.from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.05)
cent.plot();
```



Hazard footprint

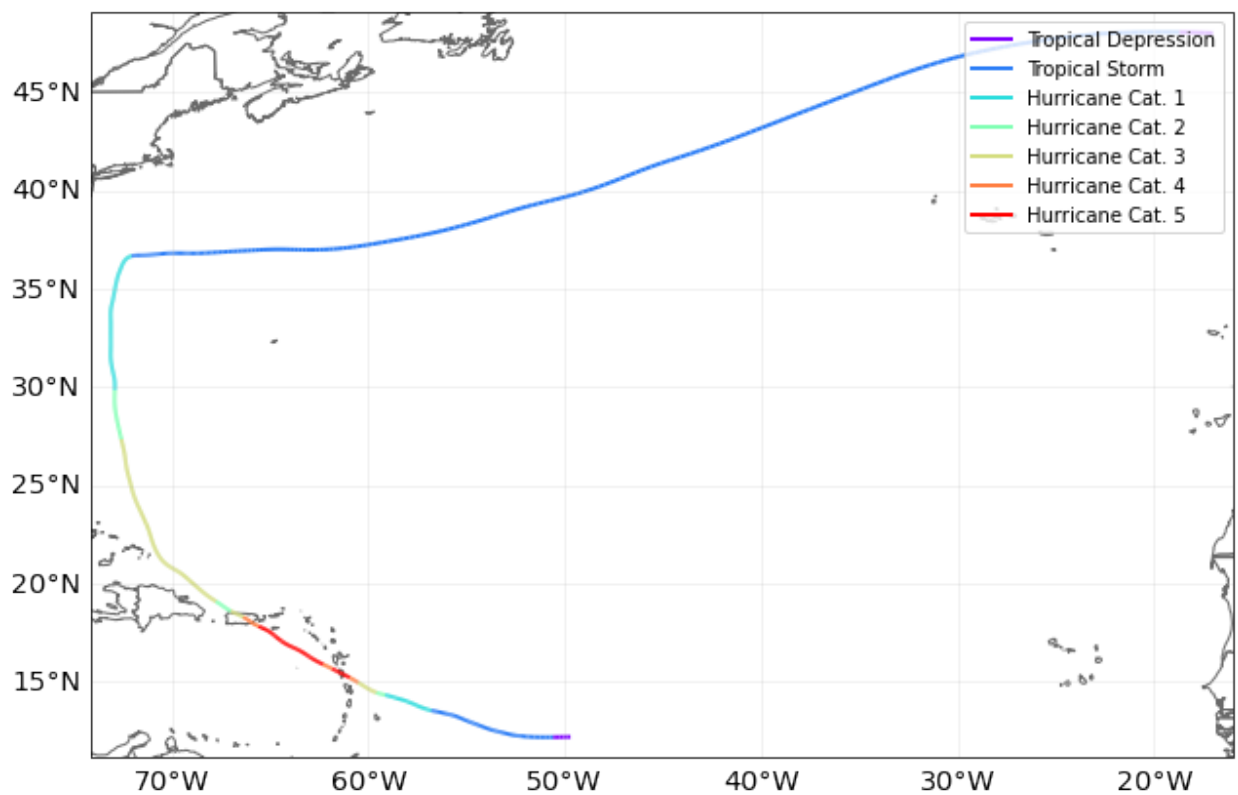
Now we're ready to create our hazard object. This will be a `TropCyclone` class, which inherits from the `Hazard` class, and has the `from_tracks` constructor method to create a hazard from a `TCTracks` object at given centroids.

```
from climada.hazard import TropCyclone

haz = TropCyclone.from_tracks(tracks, centroids=cent)
haz.check()  # verifies that the necessary data for the Hazard object is correctly␣
↪provided
```

```
2022-03-21 14:35:51,458 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2022-03-21 14:35:52,496 - climada.util.coordinates - INFO - dist_to_coast: UTM 32619␣
↪(1/2)
2022-03-21 14:35:53,234 - climada.util.coordinates - INFO - dist_to_coast: UTM 32620␣
↪(2/2)
2022-03-21 14:35:53,706 - climada.hazard.trop_cyclone - INFO - Mapping 1049 tracks to␣
↪1891 coastal centroids.
2022-03-21 14:35:56,704 - climada.hazard.trop_cyclone - INFO - Progress: 10%
2022-03-21 14:36:00,561 - climada.hazard.trop_cyclone - INFO - Progress: 20%
2022-03-21 14:36:05,356 - climada.hazard.trop_cyclone - INFO - Progress: 30%
2022-03-21 14:36:09,524 - climada.hazard.trop_cyclone - INFO - Progress: 40%
2022-03-21 14:36:15,423 - climada.hazard.trop_cyclone - INFO - Progress: 50%
2022-03-21 14:36:20,307 - climada.hazard.trop_cyclone - INFO - Progress: 60%
2022-03-21 14:36:25,005 - climada.hazard.trop_cyclone - INFO - Progress: 70%
2022-03-21 14:36:30,606 - climada.hazard.trop_cyclone - INFO - Progress: 80%
2022-03-21 14:36:35,743 - climada.hazard.trop_cyclone - INFO - Progress: 90%
2022-03-21 14:36:41,322 - climada.hazard.trop_cyclone - INFO - Progress: 100%
```

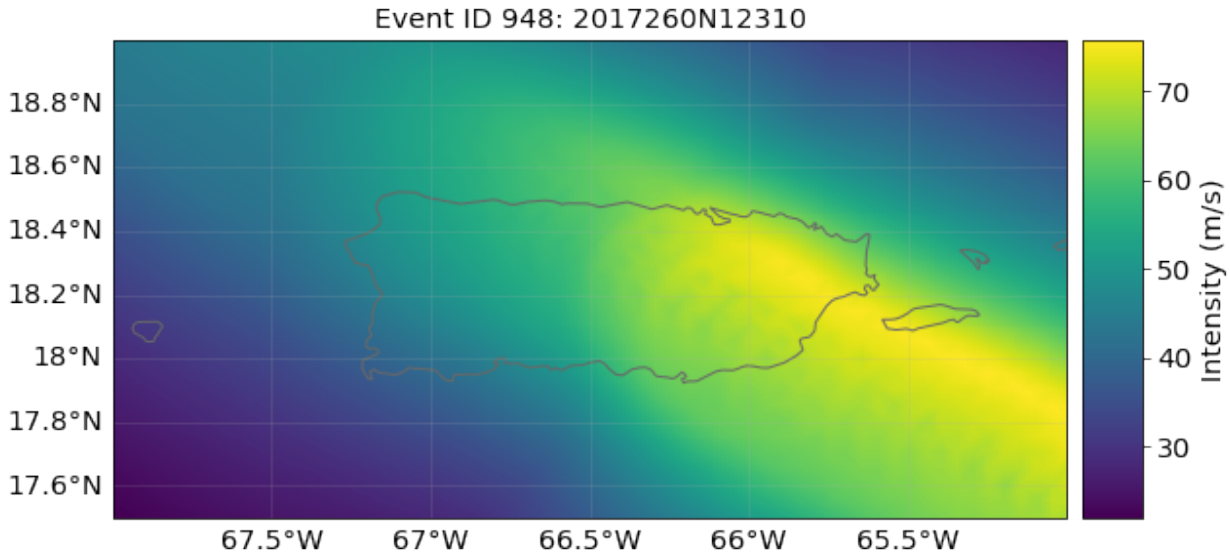
In 2017 Hurricane Maria devastated Puerto Rico. In the IBTRaCs event set, it has ID `2017260N12310` (we use this rather than the name, as IBTRaCS contains three North Atlantic storms called Maria). We can plot the track:

```
tracks.subset(
    {"sid": "2017260N12310"}
).plot();  # This is how we subset a TCTracks object
```



And plot the hazard on our centroids for Puerto Rico:

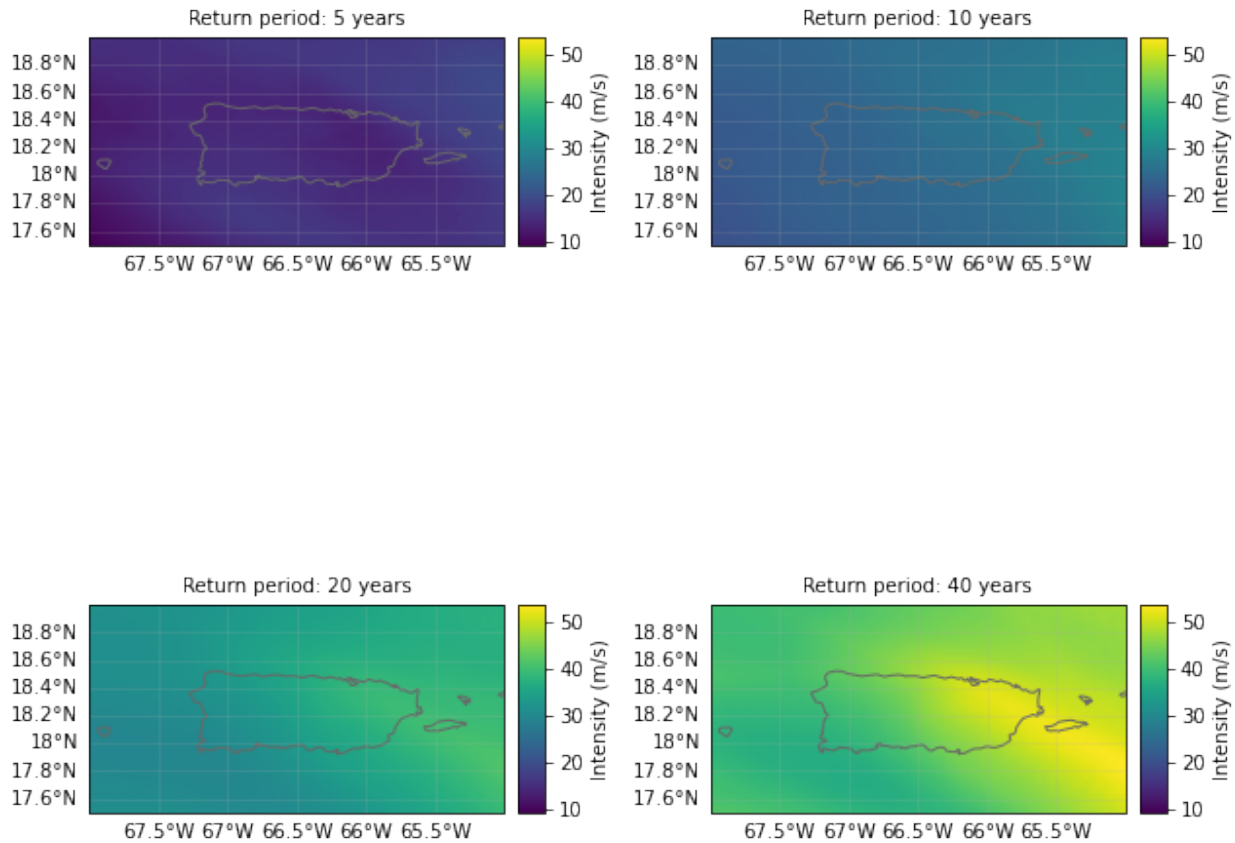
```
haz.plot_intensity(event="2017260N12310");
```



A Hazard object also lets us plot the hazard at different return periods. The IBTRaCS archive produces footprints from 1980 onwards (CLIMADA discarded earlier events) and so the historical period is short. Therefore these plots don't make sense as 'real' return periods, but we're being irresponsible and demonstrating the functionality anyway.

```
haz.plot_rp_intensity(return_periods=(5, 10, 20, 40));
```

```
2022-03-15 22:20:11,511 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [ 5 10 20 40]
```



See the *TropCyclone tutorial* for full details of the TropCyclone hazard class.

We can also recalculate event sets to reflect the effects of climate change. The `apply_climate_scenario_knu` method applies changes in intensity and frequency projected due to climate change, as described in 'Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios' (Knutson *et al.* 2015). See the *tutorial* for details.

> **Exercise:** Extend this notebook's analysis to examine the effects of climate change in Puerto Rico. You'll need to extend the historical event set with stochastic tracks to create a robust statistical storm climatology - the `TCTracks` class has the functionality to do this. Then you can apply the `apply_climate_scenario_knu` method to the generated hazard object to create a second hazard climatology representing storm activity under climate change. See how the results change using the different hazard sets.

Next we'll work on exposure and vulnerability, part of the Entity class.

2.2.5 Entity

The entity class is a container class that stores exposures and impact functions (vulnerability curves) needed for a risk calculation, and the discount rates and adaptation measures for an adaptation cost-benefit analysis.

As with Hazard objects, Entities can be read from files or created through code. The Excel template can be found in `climada_python/climada/data/system/entity_template.xlsx`.

In this tutorial we will create an Exposure object using the LitPop economic exposure module, and load a pre-defined wind damage function.

Exposures

The `Entity`'s `exposures` attribute contains geolocalized values of anything exposed to the hazard, whether monetary values of assets or number of human lives, for example. It is of type `Exposures`.

See the *Exposures tutorial* for more detail on the structure of the class, and how to create and import exposures. The *LitPop tutorial* explains how CLIMADA models economic exposures using night-time light and economic data, and is what we'll use here. To combine your exposure with OpenStreetMap's data see the OSM tutorial.

LitPop is a module that allows CLIMADA to estimate exposed populations and economic assets at any point on the planet without additional information, and in a globally consistent way. Before we try it out with the next code block, we'll need to download a data set and put it into the right folder:

1. Go to the download page on Socioeconomic Data and Applications Center (sedac).
2. You'll be asked to log in or register. Please register if you don't have an account.
3. Wait until several drop-down menus show up.
4. Choose in the drop-down menus: Temporal: single year, FileFormat: GeoTiff, Resolution: 30 seconds. Click "2020" and then "create download".
5. Copy the file "gpw_v4_population_count_rev11_2020_30_sec.tif" into the folder "~/climada/data". (Or you can run the block once to find the right path in the error message)

Now we can create an economic Exposure dataset for Puerto Rico.

```
from climada.entity.exposures import LitPop

exp_litpop = LitPop.from_countries(
    "Puerto Rico", res_arcsec=120
)  # We'll go lower resolution than default to keep it simple

exp_litpop.plot_hexbin(pop_name=True, linewidth=4, buffer=0.1);
```

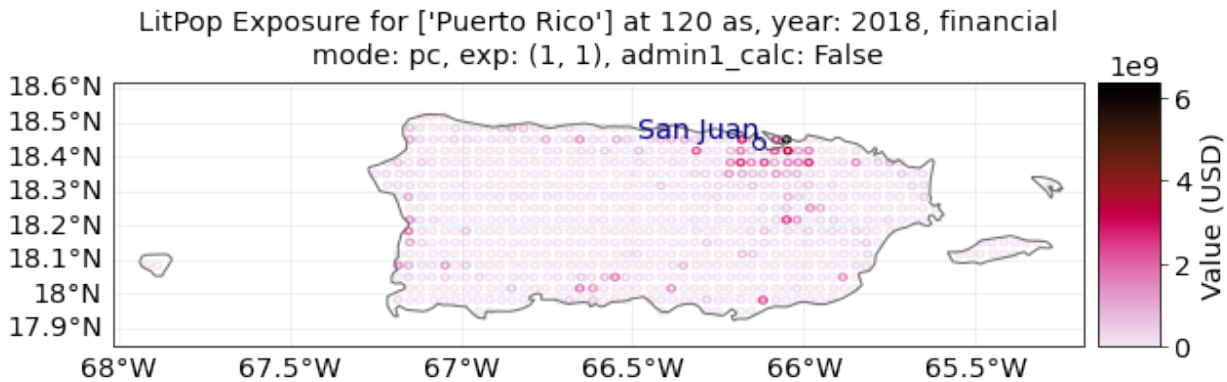
```
2022-03-21 14:37:03,770 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: PRI (630)...

2022-03-21 14:37:03,773 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2022-03-21 14:37:03,774 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-03-21 14:37:03,824 - climada.entity.exposures.litpop.nightlight - INFO - No␣
↪satellite files found locally in /home/yuyue/climada/data
2022-03-21 14:37:03,826 - climada.entity.exposures.litpop.nightlight - INFO -␣
↪Attempting to download file from https://eoimages.gsfc.nasa.gov/images/imagerecords/
↪144000/144897/BlackMarble_2016_B1_geo_gray.tif
2022-03-21 14:37:04,665 - climada.util.files_handler - INFO - Downloading https://
↪eoimages.gsfc.nasa.gov/images/imagerecords/144000/144897/BlackMarble_2016_B1_geo_
↪gray.tif to file /home/yuyue/climada/data/BlackMarble_2016_B1_geo_gray.tif
```

```
26.8kKB [00:02, 9.72kKB/s]
```

```
2022-03-21 14:37:08,919 - climada.util.files_handler - INFO - Downloading https://
↪databank.worldbank.org/data/download/Wealth-Accounts_CSV.zip to file /mnt/c/Users/
↪yyljy/Documents/climada_main/doc/tutorial/results/Wealth-Accounts_CSV.zip
```

```
1.44kKB [00:03, 429KB/s]
```

```
2022-03-21 14:37:12,440 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-03-21 14:37:13,356 - climada.util.finance - INFO - GDP PRI 2018: 1.009e+11.
2022-03-21 14:37:13,361 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-03-21 14:37:13,378 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-03-21 14:37:13,380 - climada.entity.exposures.base - INFO - category_id not set.
2022-03-21 14:37:13,381 - climada.entity.exposures.base - INFO - cover not set.
2022-03-21 14:37:13,383 - climada.entity.exposures.base - INFO - deductible not set.
2022-03-21 14:37:13,384 - climada.entity.exposures.base - INFO - centr_ not set.
2022-03-21 14:37:13,387 - climada.util.coordinates - INFO - Setting geometry points.
```



LitPop's default exposure is measured in US Dollars, with a reference year depending on the most recent data available.

Once we've created our impact function we will come back to this Exposure and give it the parameters needed to connect exposure to impacts.

Impact functions

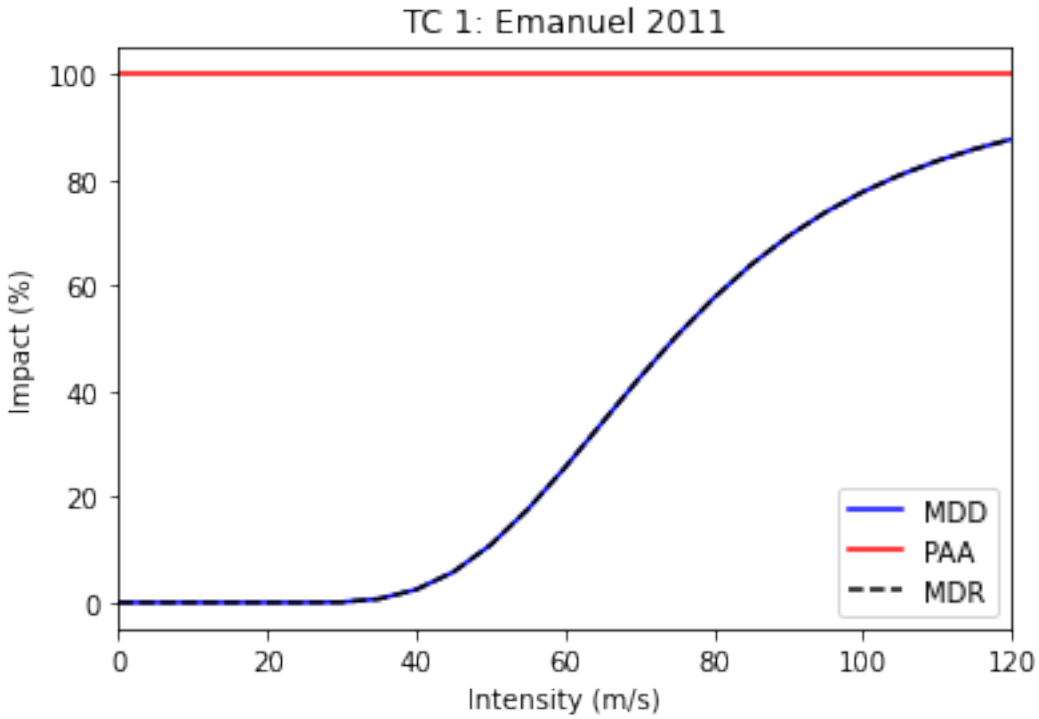
Impact functions describe a relationship between a hazard's intensity and your exposure in terms of a percentage loss. The impact is described through two terms. The Mean Degree of Damage (MDD) gives the percentage of an exposed asset's numerical value that's affected as a function of intensity, such as the damage to a building from wind in terms of its total worth. Then the Proportion of Assets Affected (PAA) gives the fraction of exposures that are affected, such as the mortality rate in a population from a heatwave. These multiply to give the Mean Damage Ratio (MDR), the average impact to an asset.

Impact functions are stored as the Entity's `impact_funcs` attribute, in an instance of the `ImpactFuncSet` class which groups one or more `ImpactFunc` objects. They can be specified manually, read from a file, or you can use CLIMADA's pre-defined impact functions. We'll use a pre-defined function for tropical storm wind damage stored in the `IFTropCyclone` class.

See the *Impact Functions tutorial* for a full guide to the class, including how data are stored and reading and writing to files.

We initialise an Impact Function with the `IFTropCyclone` class, and use its `from_emanuel_usa` method to load the Emanuel (2011) impact function. (The class also contains regional impact functions for the full globe, but we'll won't use these for now.) The class's `plot` method visualises the function, which we can see is expressed just through the Mean Degree of Damage, with all assets affected.

```
from climada.entity.impact_funcs import ImpactFuncSet, ImpfTropCyclone

imp_fun = ImpfTropCyclone.from_emanuel_usa()
imp_fun.plot();
```



The plot title also includes information about the function's ID, which were also set by the `from_emanuel_usa` class method. The hazard is "TC" and the function ID is 1. Since a study might use several impact functions - for different hazards, or for different types of exposure.

We then create an `ImpactFuncSet` object to store the impact function. This is a container class, and groups a study's impact functions together. Studies will often have several impact functions, due to multiple hazards, multiple types of exposure that are impacted differently, or different adaptation scenarios. We add it to our Entity object.

```
imp_fun_set = ImpactFuncSet([imp_fun])
```

Finally, we can update our LitPop exposure to point to the TC 1 impact function. This is done by adding a column to the exposure:

```
exp_litpop.gdf["impf_TC"] = 1
```

```
2022-03-21 14:37:53,587 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-03-21 14:37:53,591 - climada.entity.exposures.base - INFO - category_id not set.
2022-03-21 14:37:53,593 - climada.entity.exposures.base - INFO - cover not set.
2022-03-21 14:37:53,594 - climada.entity.exposures.base - INFO - deductible not set.
2022-03-21 14:37:53,595 - climada.entity.exposures.base - INFO - centr_ not set.
2022-03-21 14:37:53,600 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```

Here the `impf_TC` column tells the CLIMADA engine that for a tropical cyclone (TC) hazard, it should use the first impact function defined for TCs. We use the same impact function for all of our exposure.

This is now everything we need for a risk analysis, but while we're working on the Entity class, we can define the adaptation measures and discount rates needed for an adaptation analysis. If you're not interested in the cost-benefit analysis, you can skip ahead to the *Impact section*

Adaptation measures

CLIMADA's adaptation measures describe possible interventions that would change event hazards and impacts, and the cost of these interventions.

They are stored as `Measure` objects within a `MeasureSet` container class (similarly to `ImpactFuncSet` containing several `ImpactFunc`s), and are assigned to the `measures` attribute of the Entity.

See the *Adaptation Measures tutorial* on how to create, read and write measures. CLIMADA doesn't yet have pre-defined adaptation measures, mostly because they are hard to standardise.

The best way to understand an adaptation measure is by an example. Here's a possible measure for the creation of coastal mangroves (ignore the exact numbers, they are just for illustration):

```
from climada.entity import Measure, MeasureSet

meas_mangrove = Measure(
    name="Mangrove",
    haz_type="TC",
    color_rgb=np.array([0.2, 0.2, 0.7]),
    cost=500000000,
    mdd_impact=(1, 0),
    paa_impact=(1, -0.15),
    hazard_inten_imp=(1, -10),
)

meas_set = MeasureSet(measure_list=[meas_mangrove])
meas_set.check()
```

What values have we set here?

- The `haz_type` gives the hazard that this measure affects.
- The `cost` is a flat price that will be used in cost-benefit analyses.
- The `mdd_impact`, `paa_impact`, and `hazard_inten_imp` attributes are all tuples that describes a linear transformation to event hazard, the impact function's mean damage degree and the impact function's proportion of assets affected. The tuple `(a, b)` describes a scalar multiplication of the function and a constant to add. So `(1, 0)` is unchanged, `(1.1, 0)` increases values by 10%, and `(1, -10)` decreases all values by 10.

So the Mangrove example above costs 50,000,000 USD, protects 15% of assets from any impact at all (`paa_impact = (1, -0.15)`) and decreases the (effective) hazard intensity by 10 m/s (`hazard_inten_imp = (1, -10)`.

We can apply these measures to our existing Exposure, Hazard and Impact functions, and plot the old and new impact functions:

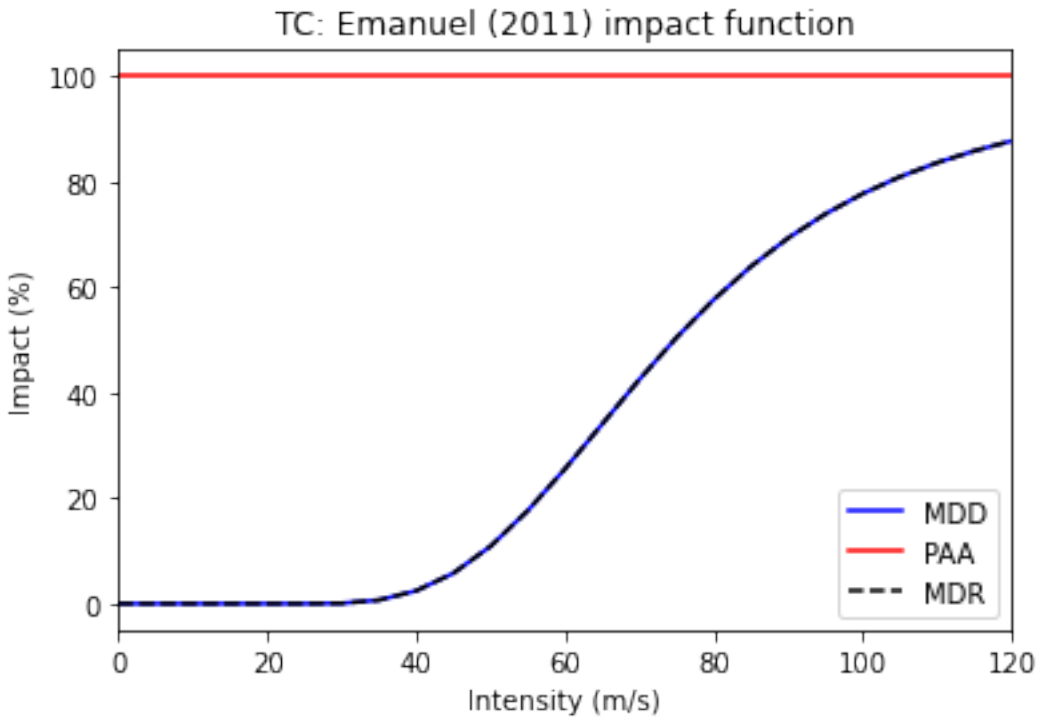
```
mangrove_exp, mangrove_imp_fun_set, mangrove_haz = meas_mangrove.apply(
    exp_litpop, imp_fun_set, haz
)
axes1 = imp_fun_set.plot()
axes1.set_title("TC: Emanuel (2011) impact function")
```

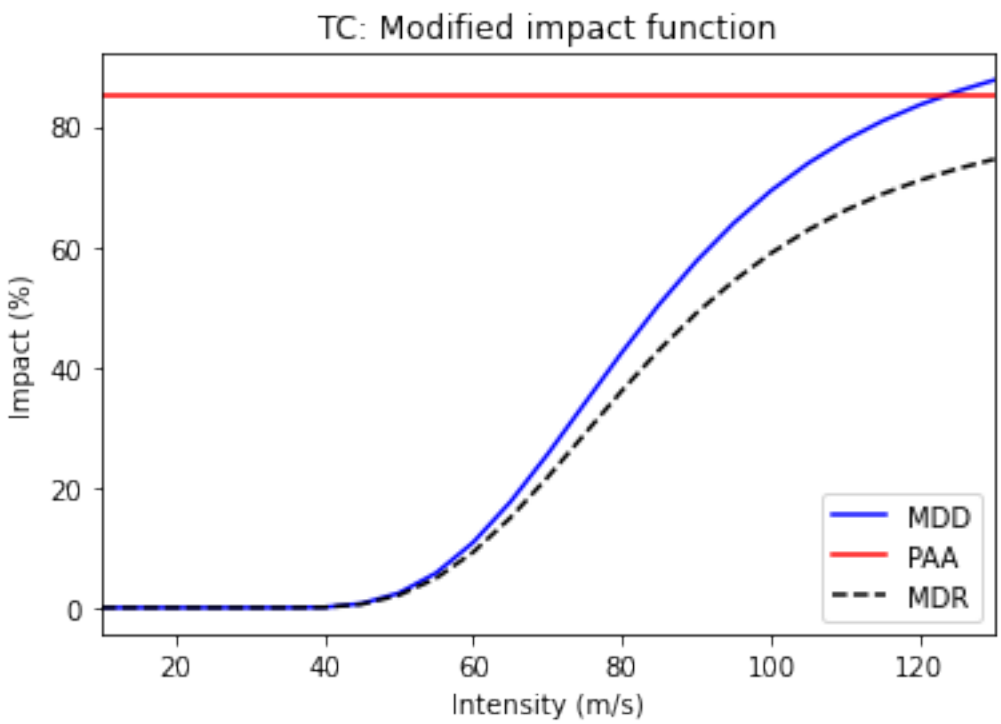
(continues on next page)

(continued from previous page)

```
axes2 = mangrove_imp_fun_set.plot()
axes2.set_title("TC: Modified impact function")
```

```
Text(0.5, 1.0, 'TC: Modified impact function')
```





Let's define a second measure. Again, the numbers here are made up, for illustration only.

```
meas_buildings = Measure(
    name="Building code",
    haz_type="TC",
    color_rgb=np.array([0.2, 0.7, 0.5]),
    cost=100000000,
    hazard_freq_cutoff=0.1,
)

meas_set.append(meas_buildings)
meas_set.check()

buildings_exp, buildings_imp_fun_set, buildings_haz = meas_buildings.apply(
    exp_litpop, imp_fun_set, haz
)
```

```
2022-03-21 14:38:24,711 - climada.entity.exposures.base - INFO - Matching 691␣
↪exposures with 1891 centroids.
2022-03-21 14:38:24,716 - climada.engine.impact - INFO - Calculating damage for 661␣
↪assets (>0) and 1049 events.
```

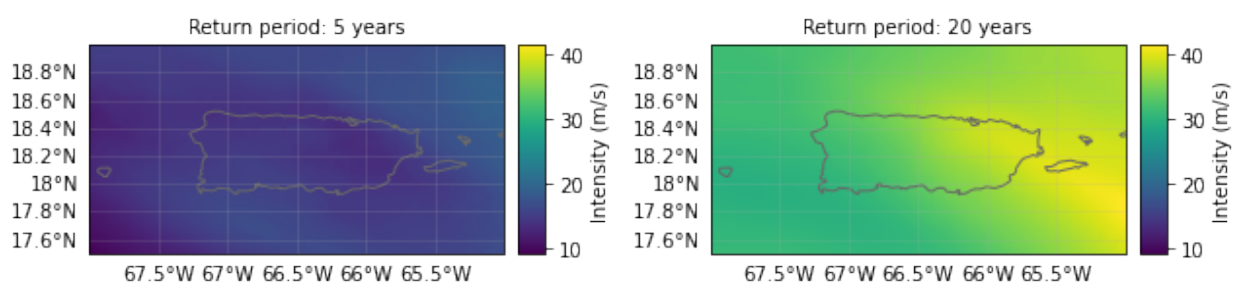
This measure describes an upgrade to building codes to withstand 10-year events. The measure costs 100,000,000 USD and, through `hazard_freq_cutoff = 0.1`, removes events with calculated impacts below the 10-year return period.

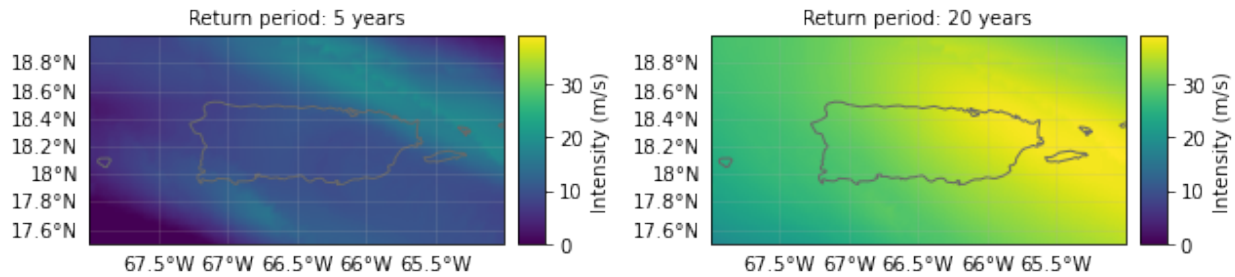
The *Adaptation Measures tutorial* describes other parameters for describing adaptation measures, including risk transfer, assigning measures to subsets of exposure, and reassigning impact functions.

We can compare the 5- and 20-year return period hazard (remember: not a real return period due to the small event set!) compared to the adjusted hazard once low-impact events are removed.

```
haz.plot_rp_intensity(return_periods=(5, 20))
buildings_haz.plot_rp_intensity(return_periods=(5, 20));
```

```
2022-03-15 22:27:56,309 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [ 5 20]
2022-03-15 22:28:13,337 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [ 5 20]
2022-03-15 22:28:13,911 - climada.hazard.base - WARNING - Exceedance intenstiy values␣
↪below 0 are set to 0.                    Reason: no negative intensity values were␣
↪found in hazard.
```





It shows there are now very few events at the 5-year return period - the new building codes removed most of these from the event set.

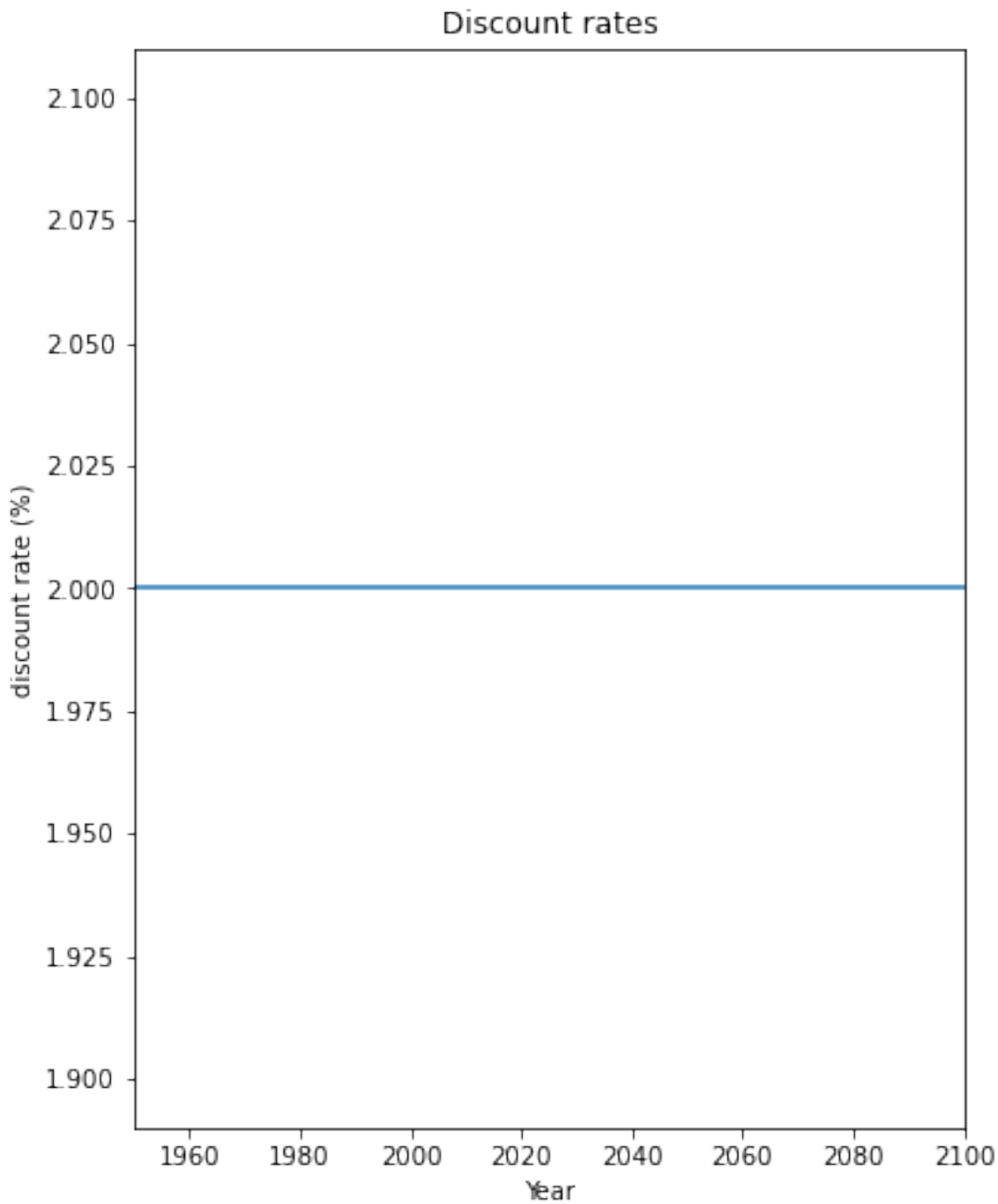
Discount rates

The `disc_rates` attribute is of type `DiscRates`. This class contains the discount rates for the following years and computes the net present value for given values.

See the *Discount Rates tutorial* for more details about creating, reading and writing the `DiscRates` class, and how it is used in calculations.

Here we will implement a simple, flat 2% discount rate.

```
from climada.entity import DiscRates

years = np.arange(1950, 2101)
rates = np.ones(years.size) * 0.02
disc = DiscRates(years=years, rates=rates)
disc.check()
disc.plot()
```



We are now ready to move to the last part of the CLIMADA model for Impact and Cost Benefit analyses.

Define Entity

We are now ready to define our Entity object that contains the exposures, impact functions, discount rates and measures.

```
from climada.entity import Entity

ent = Entity(
    exposures=exp_litpop,
    disc_rates=disc,
```

(continues on next page)

(continued from previous page)

```
    impact_func_set=imp_fun_set,
    measure_set=meas_set,
)
```

2.2.6 Engine

The CLIMADA Engine is where the main risk calculations are done. It contains two classes, `Impact`, for risk assessments, and `CostBenefit`, to evaluate adaptation measures.

Impact

Let us compute the impact of historical tropical cyclones in Puerto Rico.

Our work above has given us everything we need for a risk analysis using the Impact class. By computing the impact for each historical event, the `Impact` class provides different risk measures, as the expected annual impact per exposure, the probable maximum impact for different return periods and the total average annual impact.

Note: the configurable parameter `CONFIG.maz_matrix_size` controls the maximum matrix size contained in a chunk. You can decrease its value if you are having memory issues when using the `Impact`'s `calc` method. A high value will make the computation fast, but increase the memory use. (See the *config guide* on how to set configuration values.)

CLIMADA calculates impacts by providing exposures, impact functions and hazard to an `Impact` object's `calc` method:

```
from climada.engine import ImpactCalc

imp = ImpactCalc(ent.exposures, ent.impact_funcs, haz).impact()
```

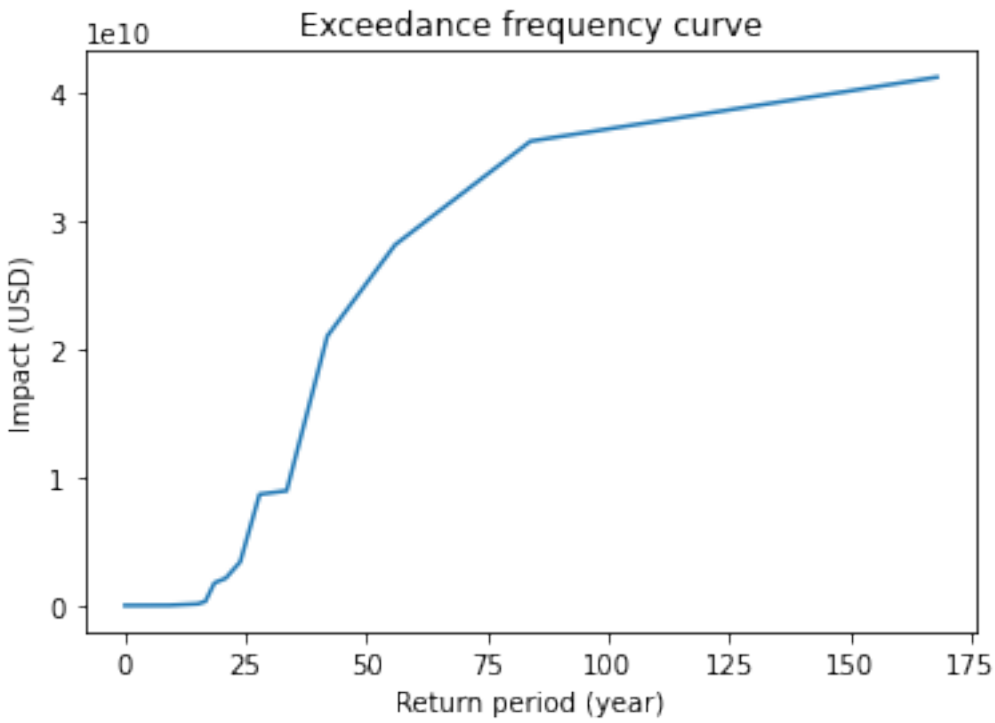
```
2022-03-21 14:38:36,337 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-21 14:38:36,343 - climada.engine.impact - INFO - Calculating damage for 661␣
↪assets (>0) and 1049 events.
```

A useful parameter for the `calc` method is `save_mat`. When set to `True` (default is `False`), the `Impact` object saves the calculated impact for each event at each point of exposure, stored as a (large) sparse matrix in the `imp_mat` attribute. This allows for more detailed analysis at the event level.

The `Impact` class includes a number of analysis tools. We can plot an exceedance frequency curve, showing us how often different damage thresholds are reached in our source data (remember this is only 40 years of storms, so not a full climatology!)

```
freq_curve = imp.calc_freq_curve()  # impact exceedance frequency curve
freq_curve.plot()

print("Expected average annual impact: {:.3e} USD".format(imp.aai_agg))
```

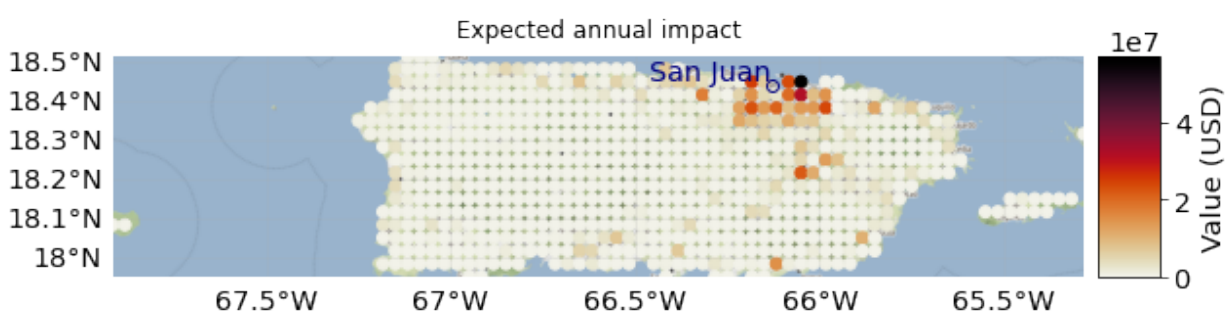
```
Expected average annual impact: 9.068e+08 USD
```



We can map the expected annual impact by exposure:

```
imp.plot_basemap_eai_exposure(buffer=0.1);  # average annual impact at each exposure
```

```
2022-03-21 14:38:43,047 - climada.util.coordinates - INFO - Setting geometry points.
2022-03-21 14:38:43,151 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.
2022-03-21 14:38:46,480 - climada.entity.exposures.base - INFO - Setting latitude and
↪longitude attributes.
```



For additional functionality, including plotting the impacts of individual events, see the *Impact tutorial*.

> **Exercise:** Plot the impacts of Hurricane Maria. To do this you'll need to set `save_mat=True` in the earlier `ImpactCalc.impact()`.

We recommend to use CLIMADA's writers in `hdf5` or `csv` whenever possible. It is also possible to save our variables in pickle format using the `save` function and load them with `load`. This will save your results in the folder specified in the configuration file. The default folder is a `results` folder which is created in the current path (see default configuration file `climada/conf/defaults.conf`). The pickle format has a *transient format* and should be avoided when possible.

```
import os
from climada.util import save, load

### Uncomment this to save - saves by default to ./results/
# save('impact_puerto_rico_tc.p', imp)

### Uncomment this to read the saved data:
# abs_path = os.path.join(os.getcwd(), 'results/impact_puerto_rico_tc.p')
# data = load(abs_path)
```

`Impact` also has `write_csv()` and `write_excel()` methods to save the impact variables, and `write_sparse_csr()` to save the impact matrix (impact per event and exposure). Use the *Impact tutorial* to get more information about these functions and the class in general.

Adaptation options appraisal

Finally, let's look at a cost-benefit analysis. The adaptation measures defined with our `Entity` can be valued by estimating their cost-benefit ratio. This is done in the class `CostBenefit`.

Let us suppose that the socioeconomic and climatoligical conditions remain the same in 2040. We then compute the cost and benefit of every adaptation measure from our Hazard and Entity (and plot them) as follows:

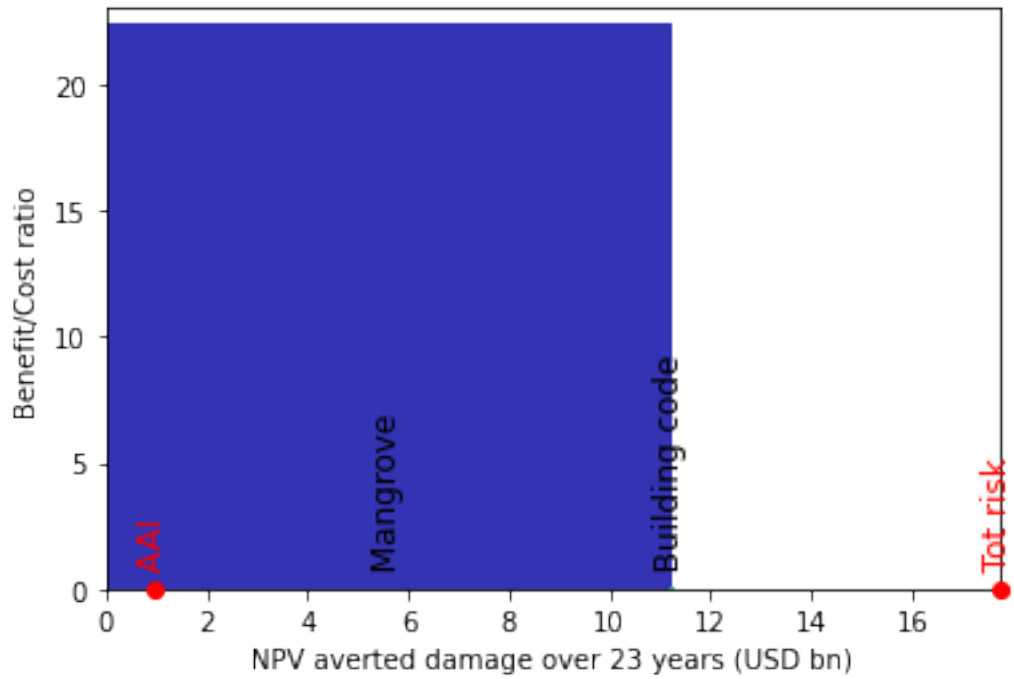
```
from climada.engine import CostBenefit

cost_ben = CostBenefit()
cost_ben.calc(haz, ent, future_year=2040)  # prints costs and benefits
cost_ben.plot_cost_benefit()
# plot cost benefit ratio and averted damage of every exposure
cost_ben.plot_event_view(
    return_per=(10, 20, 40)
);  # plot averted damage of each measure for every return period
```

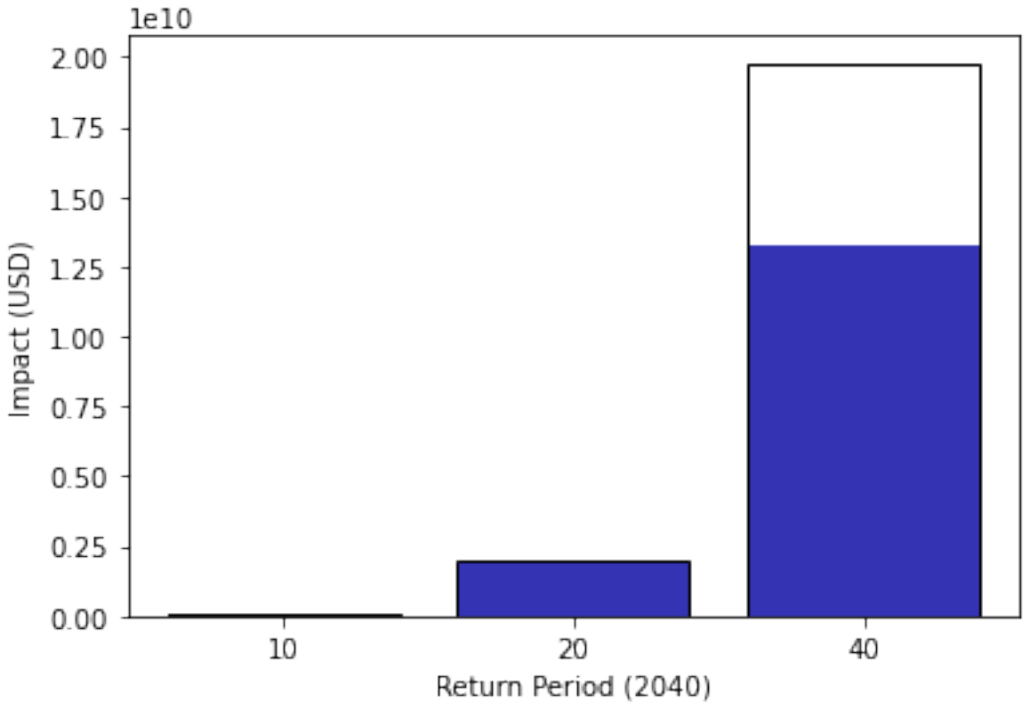
```
2022-03-15 22:32:07,393 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-15 22:32:07,397 - climada.engine.impact - INFO - Calculating damage for 691␣
↪assets (>0) and 1040 events.
2022-03-15 22:32:07,406 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-15 22:32:07,408 - climada.engine.impact - INFO - Calculating damage for 691␣
↪assets (>0) and 1040 events.
2022-03-15 22:32:07,418 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-15 22:32:07,420 - climada.engine.impact - INFO - Calculating damage for 691␣
↪assets (>0) and 1040 events.
2022-03-15 22:32:07,437 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-15 22:32:07,440 - climada.engine.impact - INFO - Calculating damage for 691␣
↪assets (>0) and 1040 events.
2022-03-15 22:32:07,452 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2040.

Measure          Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-------------  ---------------  ------------------  --------------
```

(continues on next page)

(continued from previous page)

```
Mangrove                    0.5          11.2129          22.4258
Building code               0.1           0.00761204       0.0761204

--------------------  ---------  --------
Total climate risk:   17.749     (USD bn)
Average annual risk:   0.951281  (USD bn)
Residual risk:         6.52855   (USD bn)
--------------------  ---------  --------
Net Present Values
```





This is just the start. Analyses improve as we add more adaptation measures into the mix.

Cost-benefit calculations can also include

- climate change, by specifying the `haz_future` parameter in `CostBenefit.calc()`
- changes to economic exposure over time (or to whatever exposure you're modelling) by specifying the `ent_future` parameter in `CostBenefit.calc()`
- different functions to calculate risk benefits. These are specified in `CostBenefit.calc()` and by default use changes to average annual impact
- linear, sublinear and superlinear evolution of impacts between the present and future, specified in the `imp_time_depen` parameter in `CostBenefit.calc()`

And once future hazards and exposures are defined, we can express changes to impacts over time as waterfall diagrams. See the CostBenefit class for more details.

> **Exercise:** repeat the above analysis, creating future climate hazards (see the first exercise), and future exposures based on projected economic growth. Visualise it with the `CostBenefit.plot_waterfall()` method.

2.2.7 What next?

Thanks for following this tutorial! Take time to work on the exercises it suggested, or design your own risk analysis for your own topic. More detailed tutorials for individual classes were listed in the *Features* section.

Also, explore the full CLIMADA documentation and additional resources *described at the start of this document* to learn more about CLIMADA, its structure, its existing applications and how you can contribute.

2.3 Hazard Tutorials

These guides present the *Hazard* class as well as subclasses that handle tropical cyclones and winter storms more specifically.

2.3.1 Hazard class

What is a hazard?

A hazard describes weather events such as storms, floods, droughts, or heat waves both in terms of probability of occurrence as well as physical intensity.

How are hazards embedded in the CLIMADA architecture?

Hazards are defined by the base class `Hazard` which gathers the required attributes that enable the impact computation (such as centroids, frequency per event, and intensity per event and centroid) and common methods such as readers and visualization functions. Each hazard class collects historical data or model simulations and transforms them, if necessary, in order to construct a coherent event database. Stochastic events can be generated taking into account the frequency and main intensity characteristics (such as local water depth for floods or gust speed for storms) of historical events, producing an ensemble of probabilistic events for each historical event. CLIMADA provides therefore an event-based probabilistic approach which does not depend on a hypothesis of a priori general probability distribution choices. Note that one can also reduce the probabilistic approach to a deterministic approach (e.g., story-line or forecasting) by defining the frequency to be 1. The source of the historical data (e.g. inventories or satellite images) or model simulations (e.g. synthetic tropical cyclone tracks) and the methodologies used to compute the hazard attributes and its stochastic events depend on each hazard type and are defined in its corresponding Hazard-derived class (e.g. `TropCylcone` for tropical cyclones, explained in the tutorial *TropCyclone*). This procedure provides a solid and homogeneous methodology to compute impacts worldwide. In the case where the risk analysis comprises a specific region where good quality data or models describing the hazard intensity and frequency are available, these can be directly ingested by the platform through the reader functions, skipping the hazard modelling part (in total or partially), and allowing us to easily and seamlessly combine CLIMADA with external sources. Hence the impact model can be used for a wide variety of applications, e.g. deterministically to assess the impact of a single (past or future) event or to quantify risk based on a (large) set of probabilistic events. Note that since the `Hazard` class is not an abstract class, any hazard that is not defined in CLIMADA can still be used by providing the `Hazard` attributes.

What do hazards look like in CLIMADA?

A `Hazard` contains events of some hazard type defined at `centroids`. There are certain variables in a `Hazard` instance that *are needed* to compute the impact, while others are *descriptive* and can therefore be set with default values. The full list of looks like this:

Mandatory variables	Data Type	Description
units	(str)	units of the intensity
centroids	`Centroids()`	centroids of the events
event_id	(np.array)	id (>0) of each event
frequency	(np.array)	frequency of each event in years
intensity	(sparse.csr_matrix)	intensity of the events at centroids
fraction	(sparse.csr_matrix)	fraction of affected exposures for each event at each centroid

Descriptive variables	Data Type	Description
date	(np.array	integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)
orig	(np.array	flags indicating historical events (True) or probabilistic (False)
event_name	(list(str))	name of each event (default: event_id)

Note that `intensity` and `fraction` are scipy.sparse matrices of size num_events x num_centroids. The `fraction` attribute is optional. The `Centroids` class contains the geographical coordinates where the hazard is defined. A `Centroids` instance provides the coordinates either as points or raster data together with their Coordinate Reference System (CRS). The default CRS used in climada is the usual EPSG:4326. `Centroids` provides moreover methods to compute centroids areas, on land mask, country iso mask or distance to coast.

Part 1: Read hazards from raster data

Raster data can be read in any format accepted by rasterio using `Hazard`'s `from_raster()` method. The raster information might refer to the `intensity` or `fraction`of the hazard. Different configuration options such as transforming the coordinates, changing the CRS and reading only a selected area or band are available through the `from_raster()` arguments as follows:

```
%matplotlib inline
import numpy as np
from climada.hazard import Hazard
from climada.util.constants import HAZ_DEMO_FL

# to hide the warnings
import warnings

warnings.filterwarnings("ignore")

# read intensity from raster file HAZ_DEMO_FL and set frequency for the contained␣
↪event
haz_ven = Hazard.from_raster(
    [HAZ_DEMO_FL], attrs={"frequency": np.ones(1) / 2}, haz_type="FL"
)
haz_ven.check()

# The masked values of the raster are set to 0
# Sometimes the raster file does not contain all the information, as in this case the␣
↪mask value -9999
# We mask it manuall and plot it using plot_intensity()
haz_ven.intensity[haz_ven.intensity == -9999] = 0
haz_ven.plot_intensity(
    1, smooth=False
)  # if smooth=True (default value) is used, the computation time might increase

# per default the following attributes have been set
print("event_id: ", haz_ven.event_id)
```

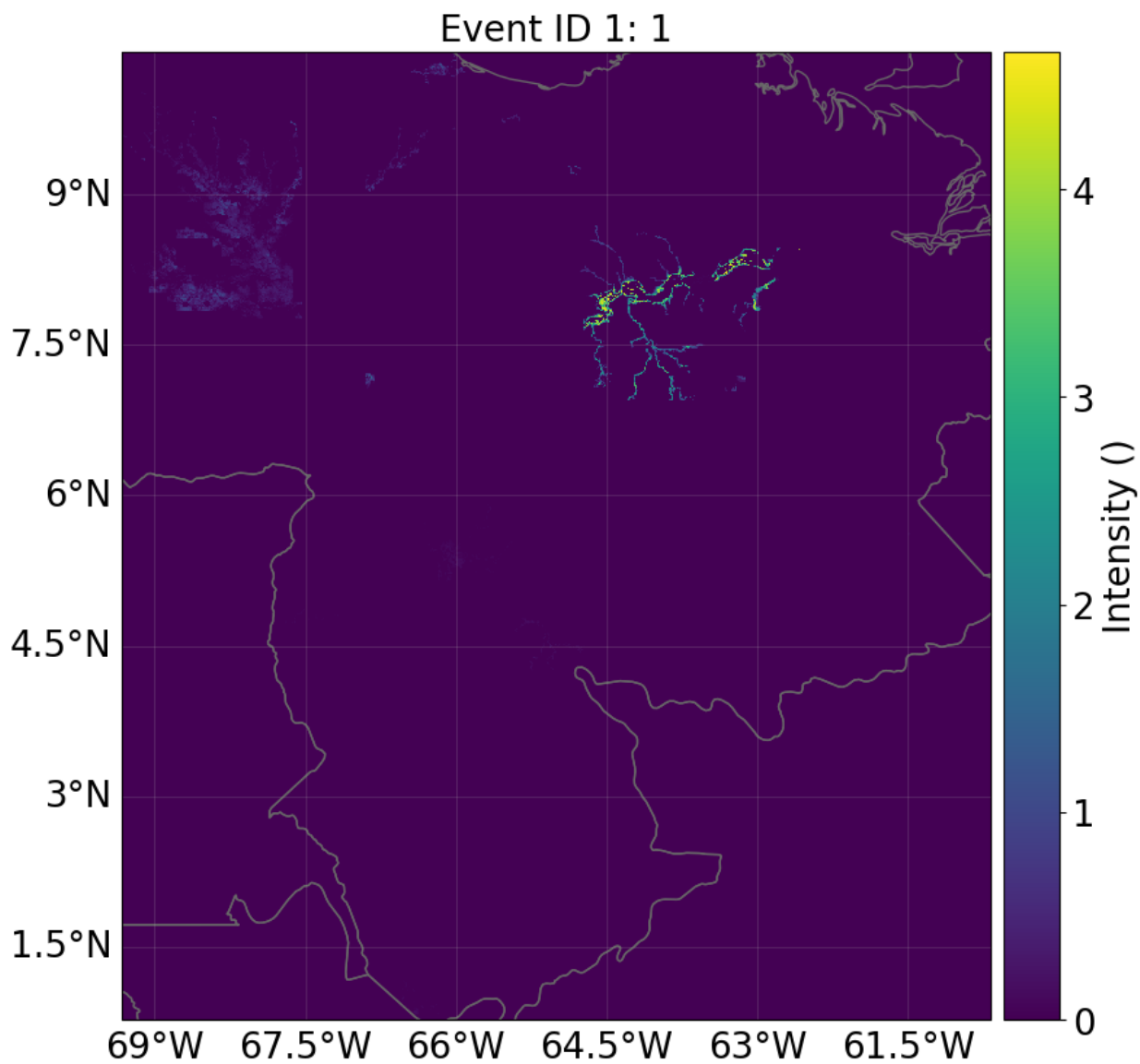
(continues on next page)

(continued from previous page)

```
print("event_name: ", haz_ven.event_name)
print("date: ", haz_ven.date)
print("frequency: ", haz_ven.frequency)
print("orig: ", haz_ven.orig)
print("min, max fraction: ", haz_ven.fraction.min(), haz_ven.fraction.max())
```

```
/Users/vgebhart/miniforge3/envs/climada_env/lib/python3.9/site-packages/dask/
↪dataframe/_pyarrow_compat.py:17: FutureWarning: Minimal version of pyarrow will␣
↪soon be increased to 14.0.1. You are using 12.0.1. Please consider upgrading.
  warnings.warn(
```

```
2024-10-01 15:52:17,203 - climada.util.coordinates - INFO - Reading /Users/vgebhart/
↪climada/demo/data/SC22000_VE__M1.grd.gz
2024-10-01 15:52:19,453 - climada.util.coordinates - INFO - Reading /Users/vgebhart/
↪climada/demo/data/SC22000_VE__M1.grd.gz
event_id:  [1]
event_name:  ['1']
date:  [1.]
frequency:  [0.5]
orig:  [ True]
min, max fraction:  0.0 1.0
```



EXERCISE:

1. Read raster data in EPSG 2201 Coordinate Reference System (CRS)
2. Read raster data in its given CRS and transform it to the affine transformation Affine(0.009000000000000341, 0.0, -69.33714959699981, 0.0, -0.009000000000000341, 10.42822096697894), height=500, width=501)
3. Read raster data in window Window(10, 10, 20, 30)

```
# Put your code here
```

```
# Solution:

# 1. The CRS can be reprojected using dst_crs option
haz = Hazard.from_raster([HAZ_DEMO_FL], dst_crs="epsg:2201", haz_type="FL")
haz.check()
```

(continues on next page)

(continued from previous page)

```
print("\n Solution 1:")
print("centroids CRS:", haz.centroids.crs)
print("raster info:", haz.centroids.get_meta())

# 2. Transformations of the coordinates can be set using the transform option and␣
↪Affine
from rasterio import Affine

haz = Hazard.from_raster(
    [HAZ_DEMO_FL],
    haz_type="FL",
    transform=Affine(
        0.009000000000000341,
        0.0,
        -69.33714959699981,
        0.0,
        -0.009000000000000341,
        10.42822096697894,
    ),
    height=500,
    width=501,
)
haz.check()
print("\n Solution 2:")
print("raster info:", haz.centroids.get_meta())
print("intensity size:", haz.intensity.shape)

# 3. A partial part of the raster can be loaded using the window or geometry
from rasterio.windows import Window

haz = Hazard.from_raster([HAZ_DEMO_FL], haz_type="FL", window=Window(10, 10, 20, 30))
haz.check()
print("\n Solution 3:")
print("raster info:", haz.centroids.get_meta())
print("intensity size:", haz.intensity.shape)
```

```
2024-10-01 15:52:28,978 - climada.util.coordinates - INFO - Reading /Users/vgebhart/
↪climada/demo/data/SC22000_VE__M1.grd.gz
2024-10-01 15:52:31,035 - climada.util.coordinates - INFO - Reading /Users/vgebhart/
↪climada/demo/data/SC22000_VE__M1.grd.gz

 Solution 1:
centroids CRS: epsg:2201
raster info: {'crs': <Projected CRS: EPSG:2201>
Name: REGVEN / UTM zone 18N
Axis Info [cartesian]:
- E[east]: Easting (metre)
- N[north]: Northing (metre)
Area of Use:
- name: Venezuela - west of 72°W.
- bounds: (-73.38, 7.02, -71.99, 11.62)
Coordinate Operation:
```

(continues on next page)

(continued from previous page)

```
- name: UTM zone 18N
- method: Transverse Mercator
Datum: Red Geodesica Venezolana
- Ellipsoid: GRS 1980
- Prime Meridian: Greenwich
, 'height': 1091, 'width': 978, 'transform': Affine(1011.5372910988517, 0.0, 1120744.
↪5486664253,
       0.0, -1011.5372910988517, 1189133.7652687666)}
2024-10-01 15:52:32,933 - climada.util.coordinates - INFO - Reading /Users/vgebhart/
↪climada/demo/data/SC22000_VE__M1.grd.gz
2024-10-01 15:52:34,619 - climada.util.coordinates - INFO - Reading /Users/vgebhart/
↪climada/demo/data/SC22000_VE__M1.grd.gz

 Solution 2:
raster info: {'crs': <Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
, 'height': 500, 'width': 501, 'transform': Affine(0.009000000000000341, 0.0, -69.
↪33714959699981,
       0.0, -0.009000000000000341, 10.42822096697894)}
intensity size: (1, 250500)
2024-10-01 15:52:36,437 - climada.util.coordinates - INFO - Reading /Users/vgebhart/
↪climada/demo/data/SC22000_VE__M1.grd.gz
2024-10-01 15:52:36,453 - climada.util.coordinates - INFO - Reading /Users/vgebhart/
↪climada/demo/data/SC22000_VE__M1.grd.gz

 Solution 3:
raster info: {'crs': <Geographic 2D CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
, 'height': 30, 'width': 20, 'transform': Affine(0.009000000000000341, 0.0, -69.
↪2471495969998,
       0.0, -0.009000000000000341, 10.338220966978936)}
intensity size: (1, 600)
```

Part 2: Read hazards from other data

- excel: Hazards can be read from Excel files following the template in `climada_python/climada/data/system/hazard_template.xlsx` using the `from_excel()` method.
- MATLAB: Hazards generated with CLIMADA's MATLAB version (.mat format) can be read using `from_mat()`.
- vector data: Use `Hazard`'s `from_vector`-constructor to read shape data (all formats supported by fiona).
- hdf5: Hazards generated with the CLIMADA in Python (.h5 format) can be read using `from_hdf5()`.

```
from climada.hazard import Hazard, Centroids
from climada.util import HAZ_DEMO_H5  # CLIMADA's Python file

# Hazard needs to know the acronym of the hazard type to be constructed!!! Use 'NA'␣
↪if not known.
haz_tc_fl = Hazard.from_hdf5(
    HAZ_DEMO_H5
)  # Historic tropical cyclones in Florida from 1990 to 2004
haz_tc_fl.check()  # Use always the check() method to see if the hazard has been␣
↪loaded correctly
```

```
2024-10-01 15:52:36,511 - climada.hazard.io - INFO - Reading /Users/vgebhart/climada/
↪demo/data/tc_fl_1990_2004.h5
```

Part 3: Define hazards manually

A `Hazard` can be defined by filling its values one by one, as follows:

```
# setting points
import numpy as np
from scipy import sparse

lat = np.array(
    [
        26.933899,
        26.957203,
        26.783846,
        26.645524,
        26.897796,
        26.925359,
        26.914768,
        26.853491,
        26.845099,
        26.82651,
        26.842772,
        26.825905,
        26.80465,
        26.788649,
        26.704277,
        26.71005,
        26.755412,
        26.678449,
```

(continues on next page)

(continued from previous page)

```
        26.725649,
        26.720599,
        26.71255,
        26.6649,
        26.664699,
        26.663149,
        26.66875,
        26.638517,
        26.59309,
        26.617449,
        26.620079,
        26.596795,
        26.577049,
        26.524585,
        26.524158,
        26.523737,
        26.520284,
        26.547349,
        26.463399,
        26.45905,
        26.45558,
        26.453699,
        26.449999,
        26.397299,
        26.4084,
        26.40875,
        26.379113,
        26.3809,
        26.349068,
        26.346349,
        26.348015,
        26.347957,
    ]
)

lon = np.array(
    [
        -80.128799,
        -80.098284,
        -80.748947,
        -80.550704,
        -80.596929,
        -80.220966,
        -80.07466,
        -80.190281,
        -80.083904,
        -80.213493,
        -80.0591,
        -80.630096,
        -80.075301,
        -80.069885,
        -80.656841,
```

(continues on next page)

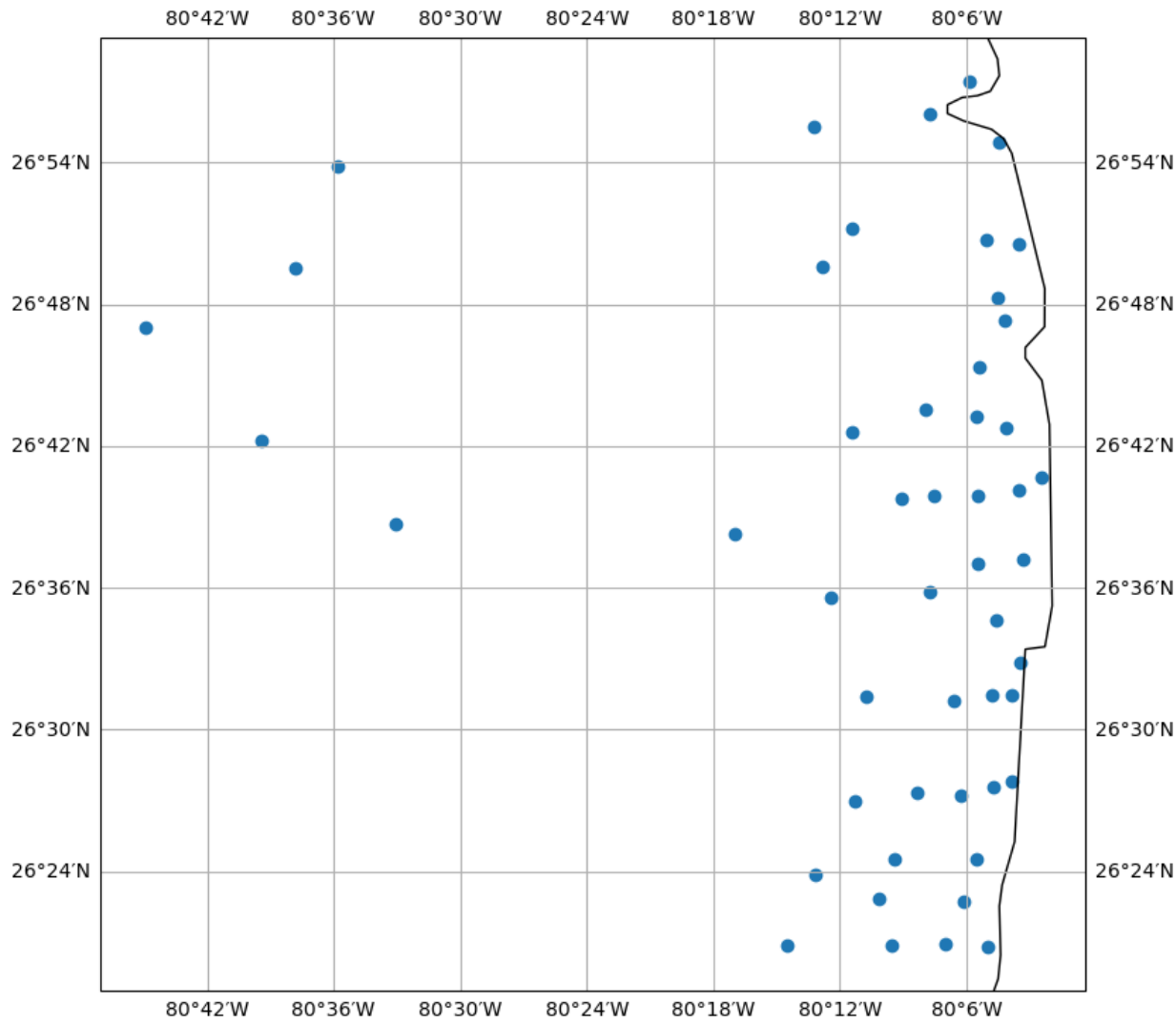
(continued from previous page)

```
        -80.190085,
        -80.08955,
        -80.041179,
        -80.1324,
        -80.091746,
        -80.068579,
        -80.090698,
        -80.1254,
        -80.151401,
        -80.058749,
        -80.283371,
        -80.206901,
        -80.090649,
        -80.055001,
        -80.128711,
        -80.076435,
        -80.080105,
        -80.06398,
        -80.178973,
        -80.110519,
        -80.057701,
        -80.064251,
        -80.07875,
        -80.139247,
        -80.104316,
        -80.188545,
        -80.21902,
        -80.092391,
        -80.1575,
        -80.102028,
        -80.16885,
        -80.116401,
        -80.08385,
        -80.241305,
        -80.158855,
    ]
)

n_cen = lon.size  # number of centroids
n_ev = 10  # number of events

intensity = sparse.csr_matrix(np.random.random((n_ev, n_cen)))
fraction = intensity.copy()
fraction.data.fill(1)

haz = Hazard(
    haz_type="TC",
    intensity=intensity,
    fraction=fraction,
    centroids=Centroids(lat=lat, lon=lon),  # default crs used
    units="m",
    event_id=np.arange(n_ev, dtype=int),
```

(continues on next page)

(continued from previous page)

```
    event_name=[
        "ev_12",
        "ev_21",
        "Maria",
        "ev_35",
        "Irma",
        "ev_16",
        "ev_15",
        "Edgar",
        "ev_1",
        "ev_9",
    ],
    date=np.array(
        [721166, 734447, 734447, 734447, 721167, 721166, 721167, 721200, 721166,␣
↪721166]
    ),
    orig=np.zeros(n_ev, bool),
    frequency=np.ones(n_ev) / n_ev,
)

haz.check()
haz.centroids.plot();
```



Or the `Hazard` can be defined with a grid:

```
# using from_pnt_bounds

# bounds
left, bottom, right, top = (
    -72,
    -3.0,
    -52.0,
    22,
)  # the bounds refer to the bounds of the center of the pixel
# resolution
res = 0.5
centroids = Centroids.from_pnt_bounds(
    (left, bottom, right, top), res
)  # default crs used
```

```
# the same can be done with the method `from_meta`, by definition of a raster meta␣
```

(continues on next page)

(continued from previous page)

```
↪object

import rasterio
from climada.util.constants import DEF_CRS

# raster info:
# border upper left corner (of the pixel, not of the center of the pixel)
max_lat = top + res / 2
min_lon = left - res / 2
# resolution in lat and lon
d_lat = -res  # negative because starting in upper corner
d_lon = res  # same step as d_lat
# number of points
n_lat, n_lon = centroids.shape

# meta: raster specification
meta = {
    "dtype": "float32",
    "width": n_lon,
    "height": n_lat,
    "crs": DEF_CRS,
    "transform": rasterio.Affine(a=d_lon, b=0.0, c=min_lon, d=0.0, e=d_lat, f=max_
↪lat),
}

centroids_from_meta = Centroids.from_meta(meta)  # default crs used

centroids_from_meta == centroids
```

```
True
```

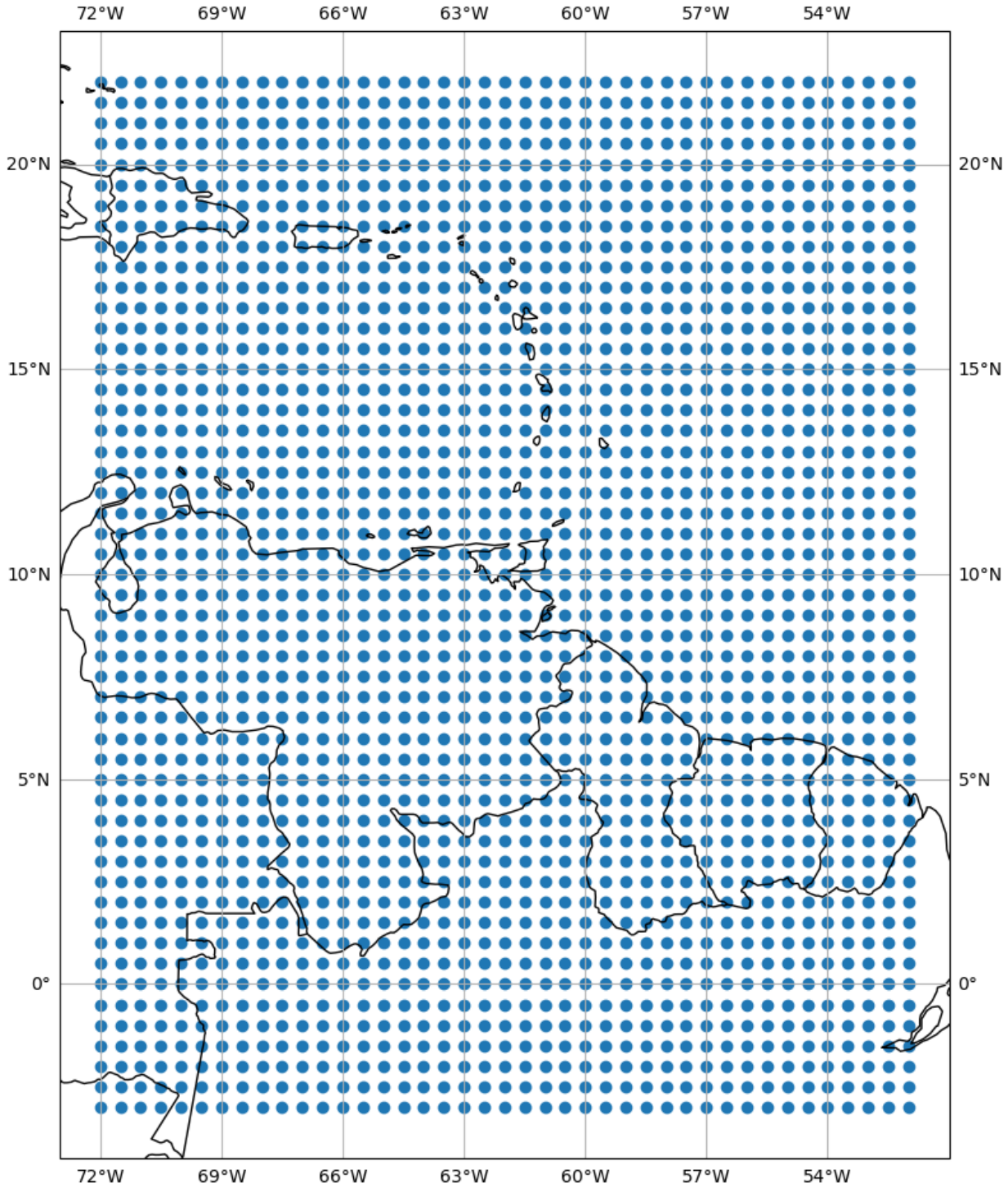
```
# create a Hazard object with random events

import numpy as np
from scipy import sparse

n_ev = 10  # number of events

intensity = sparse.csr_matrix(np.random.random((n_ev, centroids.size)))
fraction = intensity.copy()
fraction.data.fill(1)

haz = Hazard(
    "TC",
    centroids=centroids,
    intensity=intensity,
    fraction=fraction,
    units="m",
    event_id=np.arange(n_ev, dtype=int),
    event_name=[
        "ev_12",
```

(continues on next page)

(continued from previous page)

```
        "ev_21",
        "Maria",
        "ev_35",
        "Irma",
        "ev_16",
        "ev_15",
        "Edgar",
        "ev_1",
        "ev_9",
    ],
    date=np.array(
        [721166, 734447, 734447, 734447, 721167, 721166, 721167, 721200, 721166,␣
↪721166]
    ),
    orig=np.zeros(n_ev, bool),
    frequency=np.ones(n_ev) / n_ev,
)

haz.check()
print("Check centroids borders:", haz.centroids.total_bounds)
haz.centroids.plot();
```

```
Check centroids borders: [-72.  -3. -52.  22.]
```


72°W
69°W
66°W
63°W
60°W
57°W
54°W
20°N
15°N
10°N
5°N
0°

Part 4: Analyse Hazards

The following methods can be used to analyse the data in `Hazard`:

- `calc_year_set()` method returns a dictionary with all the historical (not synthetic) event ids that happened at each year.
- `get_event_date()` returns strings of dates in ISO format.
- To obtain the relation between event ids and event names, two methods can be used `get_event_name()` and `get_event_id()`.

Other methods to handle one or several Hazards are:

- the property `size` returns the number of events contained.
- `append()` is used to expand events with data from another `Hazard` (and same centroids).
- `select()` returns a new hazard with the selected region, date and/or synthetic or historical filter.
- `remove_duplicates()` removes events with same name and date.
- `local_exceedance_intensity()` returns a gdf.GeoDataFrame with the exceedence frequency at every frequency and provided return periods.
- `local_return_period()` returns a gdf.GeoDataFrame with the return periods at every centroid for user-specified threshold intensities.
- `reproject_vector()` is a method to change the centroids' CRS.

`Centroids` methods:

- centroids properties such as area per pixel, distance to coast, country ISO code, on land mask or elevation are available through different `set_XX()` methods.
- `set_lat_lon_to_meta()` computes the raster `meta` dictionary from present lat and lon. `set_meta_to_lat_lon()` computes lat and lon of the center of the pixels described in attribute `meta`. The raster `meta` information contains at least: `width`, `height`, `crs` and `transform` data (use help(Centroids) for more info). Using raster centroids can increase computing performance for several computations.
- when using lats and lons (vector data) the `geopandas.GeoSeries geometry` attribute contains the CRS information and can be filled with point shapes to perform different computation. The geometry points can be then released using `empty_geometry_points()`.

EXERCISE:

Using the previous hazard `haz_tc_fl` answer these questions:

1. How many synthetic events are contained?
2. Generate a hazard with historical hurricanes ocurring between 1995 and 2001.
3. How many historical hurricanes occured in 1999? Which was the year with most hurricanes between 1995 and 2001?
4. What is the number of centroids with distance to coast smaller than 1km?

```
# Put your code here:
```

```
# help(hist_tc.centroids) # If you want to run it, do it after you execute the next␣
↪block
```

```
# SOLUTION:

# 1.How many synthetic events are contained?
print("Number of total events:", haz_tc_fl.size)
print("Number of synthetic events:", np.logical_not(haz_tc_fl.orig).astype(int).sum())

# 2. Generate a hazard with historical hurricanes ocurring between 1995 and 2001.
hist_tc = haz_tc_fl.select(date=("1995-01-01", "2001-12-31"), orig=True)
print("Number of historical events between 1995 and 2001:", hist_tc.size)

# 3. How many historical hurricanes occured in 1999? Which was the year with most␣
↪hurricanes between 1995 and 2001?
ev_per_year = hist_tc.calc_year_set()  # events ids per year
print("Number of events in 1999:", ev_per_year[1999].size)
max_year = 1995
max_ev = ev_per_year[1995].size
for year, ev in ev_per_year.items():
    if ev.size > max_ev:
        max_year = year
print("Year with most hurricanes between 1995 and 2001:", max_year)

# 4. What is the number of centroids with distance to coast smaller than 1km?
num_cen_coast = np.argwhere(hist_tc.centroids.get_dist_coast() < 1000).size
print("Number of centroids close to coast: ", num_cen_coast)
```

```
Number of total events: 216
Number of synthetic events: 0
Number of historical events between 1995 and 2001: 109
Number of events in 1999: 16
Year with most hurricanes between 1995 and 2001: 1995
2024-10-01 15:52:37,708 - climada.util.coordinates - INFO - Sampling from /Users/
↪vgebhart/climada/data/GMT_intermediate_coast_distance_01d.tif
Number of centroids close to coast:  67
```

Part 5: Visualize Hazards

There are three different plot functions: `plot_intensity()`, `plot_fraction()`, and `plot_rp_intensity()`. Depending on the inputs, different properties can be visualized. Check the documentation of the functions. Using the function `local_return_period()` and the util function `plot_from_gdf()`, one can plot local return periods for specific hazard intensities.

```
help(haz_tc_fl.plot_intensity)
help(haz_tc_fl.plot_rp_intensity)
```

```
Help on method plot_intensity in module climada.hazard.plot:

plot_intensity(event=None, centr=None, smooth=True, axis=None, adapt_fontsize=True,␣
↪**kwargs) method of climada.hazard.base.Hazard instance
    Plot intensity values for a selected event or centroid.

    Parameters
```

(continues on next page)

(continued from previous page)

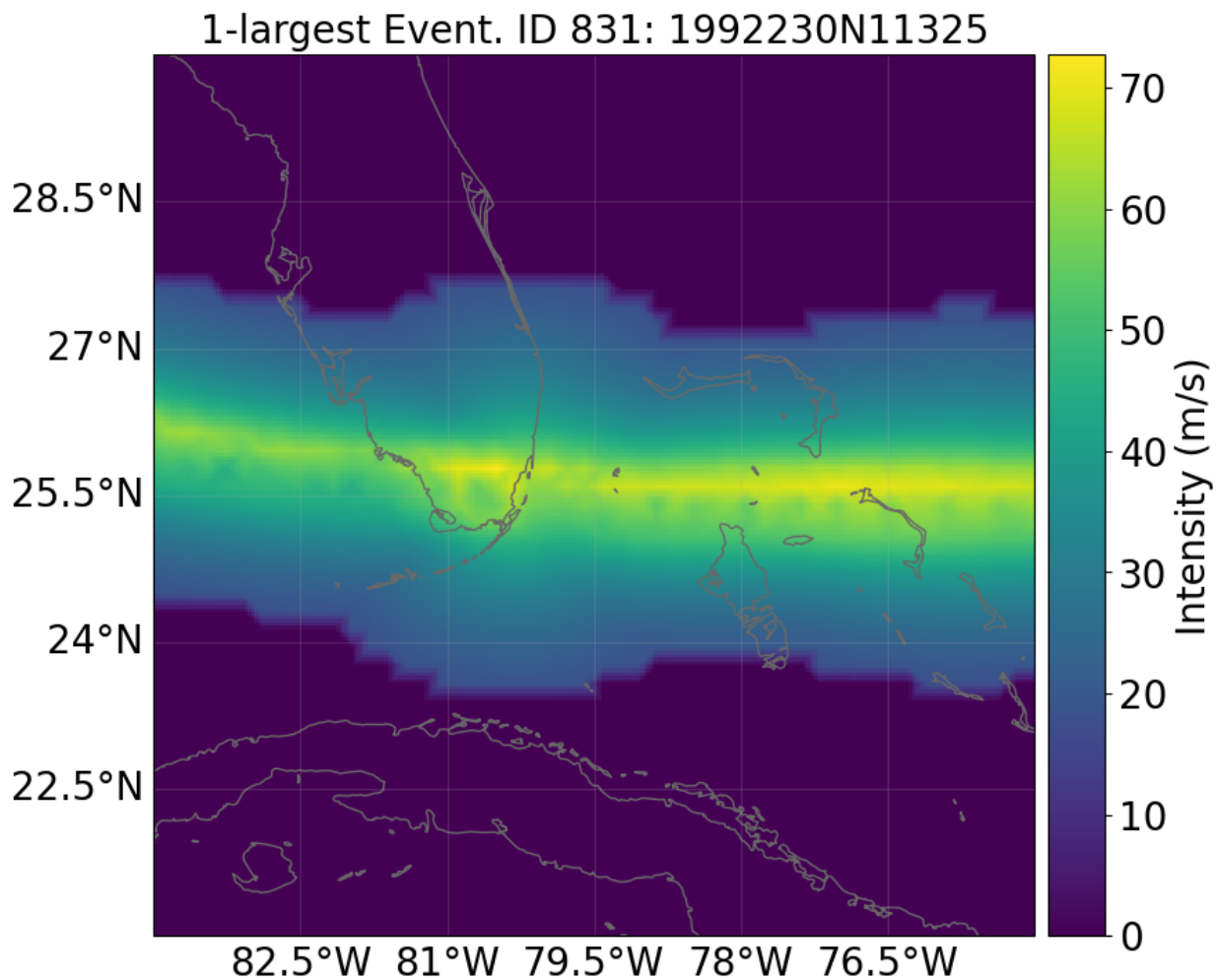
```
    ----------
    event: int or str, optional
        If event > 0, plot intensities of
        event with id = event. If event = 0, plot maximum intensity in
        each centroid. If event < 0, plot abs(event)-largest event. If
        event is string, plot events with that name.
    centr: int or tuple, optional
        If centr > 0, plot intensity
        of all events at centroid with id = centr. If centr = 0,
        plot maximum intensity of each event. If centr < 0,
        plot abs(centr)-largest centroid where higher intensities
        are reached. If tuple with (lat, lon) plot intensity of nearest
        centroid.
    smooth: bool, optional
        Rescale data to RESOLUTIONxRESOLUTION pixels (see constant
        in module `climada.util.plot`)
    axis: matplotlib.axes._subplots.AxesSubplot, optional
        axis to use
    kwargs: optional
        arguments for pcolormesh matplotlib function
        used in event plots or for plot function used in centroids plots

    Returns
    -------
        matplotlib.axes._subplots.AxesSubplot

    Raises
    ------
        ValueError

Help on method plot_rp_intensity in module climada.hazard.plot:

plot_rp_intensity(return_periods=(25, 50, 100, 250), smooth=True, axis=None,␣
↪figsize=(9, 13), adapt_fontsize=True, **kwargs) method of climada.hazard.base.
↪Hazard instance
    This function is deprecated,
    use Impact.local_exceedance_impact and util.plot.plot_from_gdf instead.
```

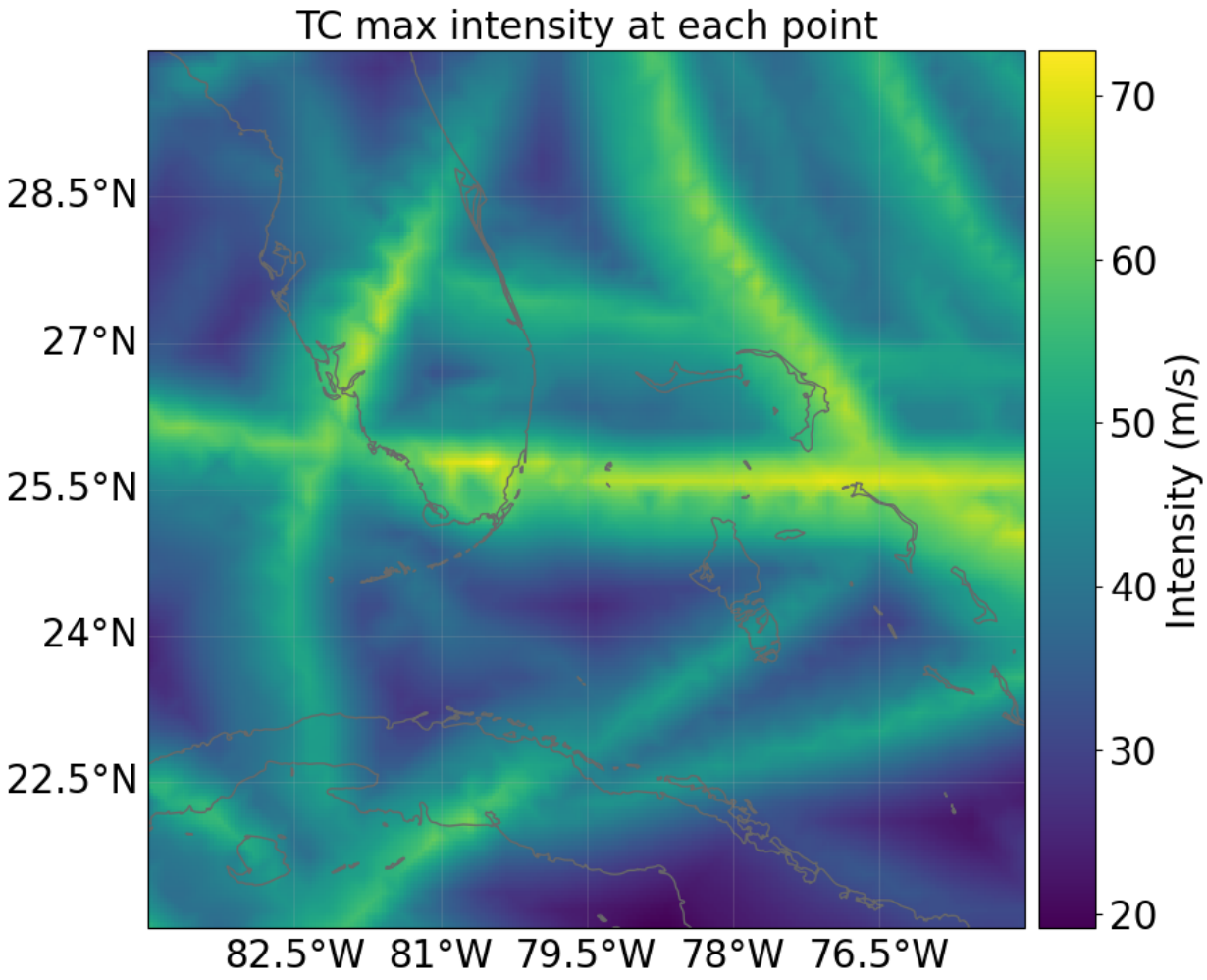
```
# 1. intensities of the largest event (defined as greater sum of intensities):
# all events:
haz_tc_fl.plot_intensity(
    event=-1
)  # largest historical event: 1992230N11325 hurricane ANDREW

# 2. maximum intensities at each centroid:
haz_tc_fl.plot_intensity(event=0)

# 3. intensities of hurricane 1998295N12284:
haz_tc_fl.plot_intensity(event="1998295N12284", cmap="BuGn")  # setting color map

# 4. tropical cyclone intensities maps for the return periods [10, 50, 75, 100]
exceedance_intensities, label, column_label = haz_tc_fl.local_exceedance_intensity(
```

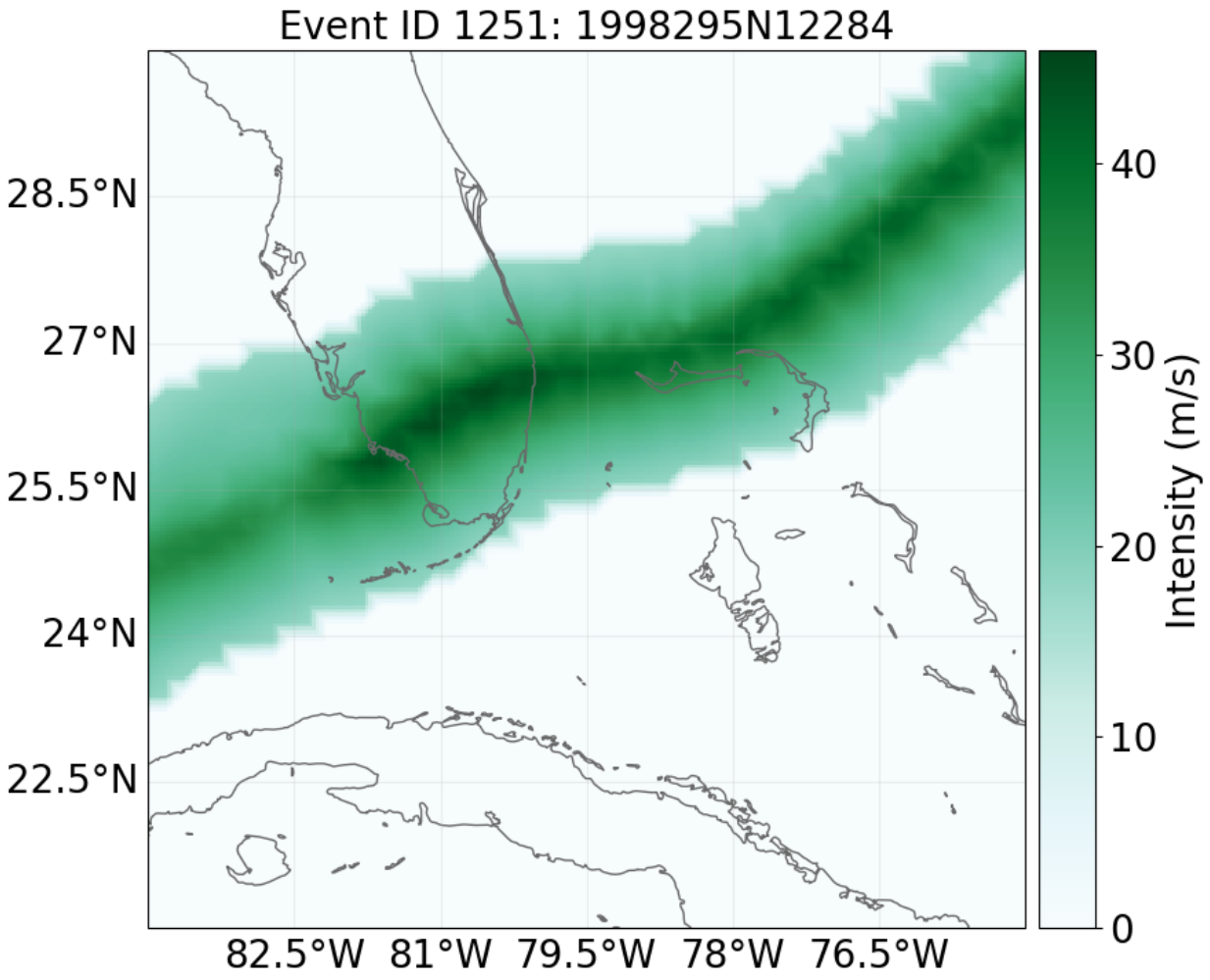
(continues on next page)

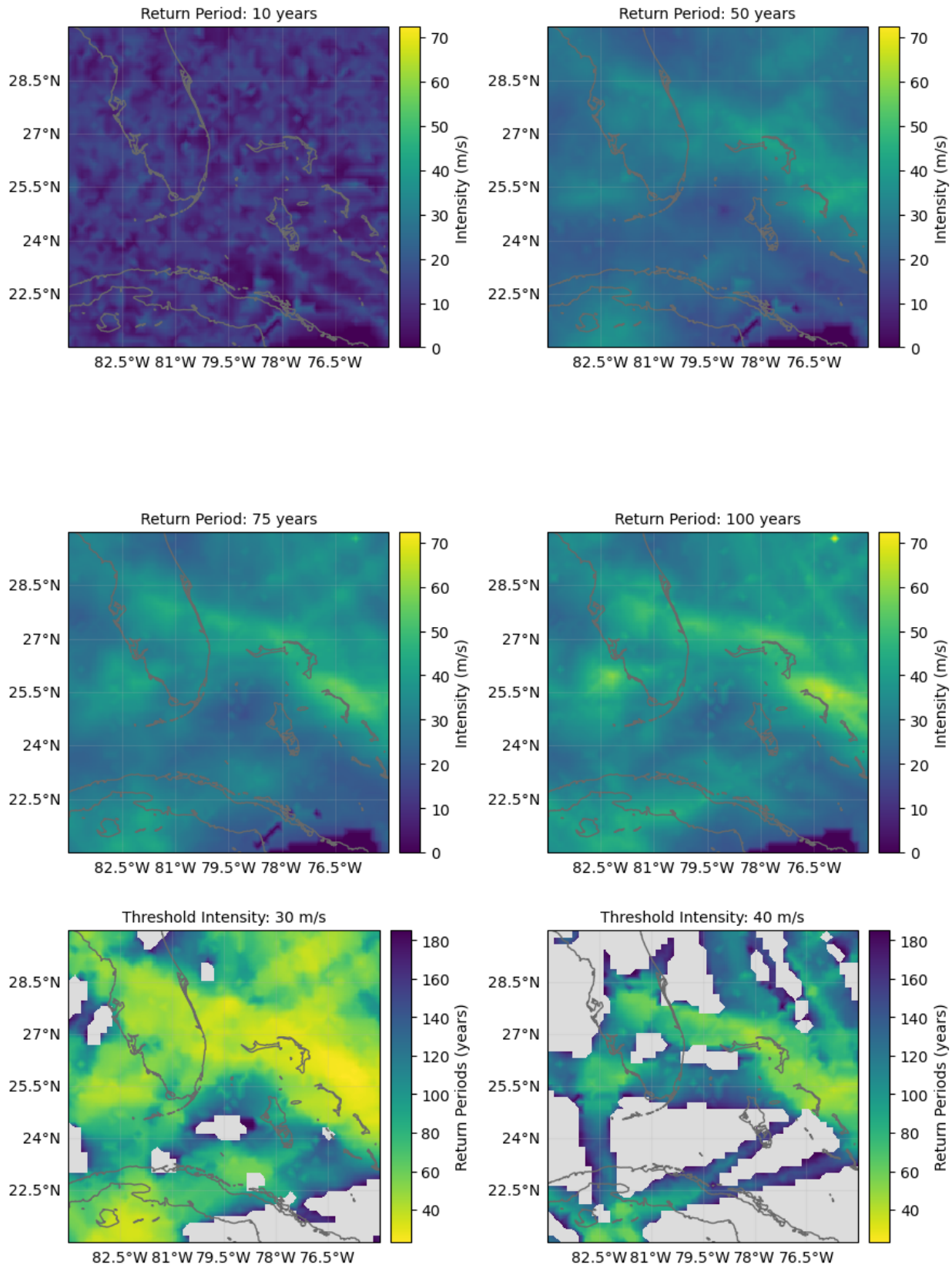
(continued from previous page)

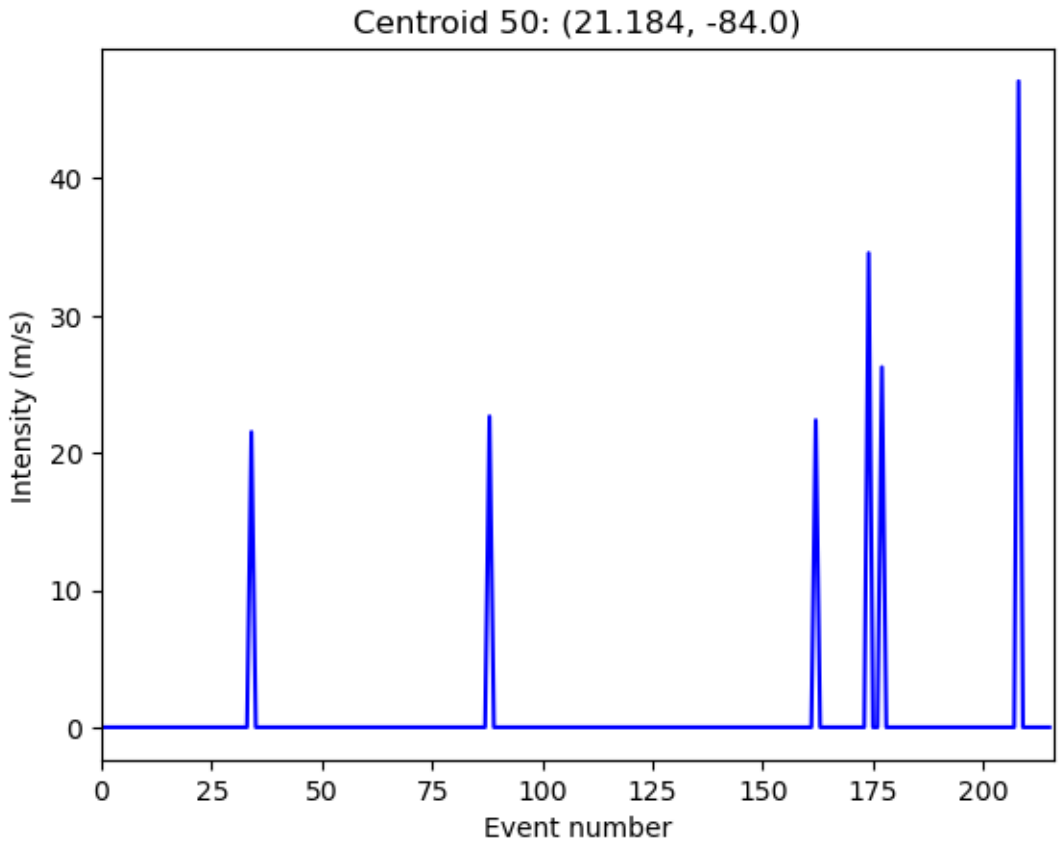
```
    [10, 50, 75, 100], method="extrapolate"
)
from climada.util.plot import plot_from_gdf

plot_from_gdf(exceedance_intensities, colorbar_name=label, title_subplots=column_
↪label)

# 5. tropical cyclone return period maps for the threshold intensities [30, 40]
return_periods, label, column_label = haz_tc_fl.local_return_period([30, 40])
from climada.util.plot import plot_from_gdf

plot_from_gdf(return_periods, colorbar_name=label, title_subplots=column_label)

# 6. intensities of all the events in centroid with id 50
haz_tc_fl.plot_intensity(centr=50)

# 7. intensities of all the events in centroid closest to lat, lon = (26.5, -81)
haz_tc_fl.plot_intensity(centr=(26.5, -81));
```

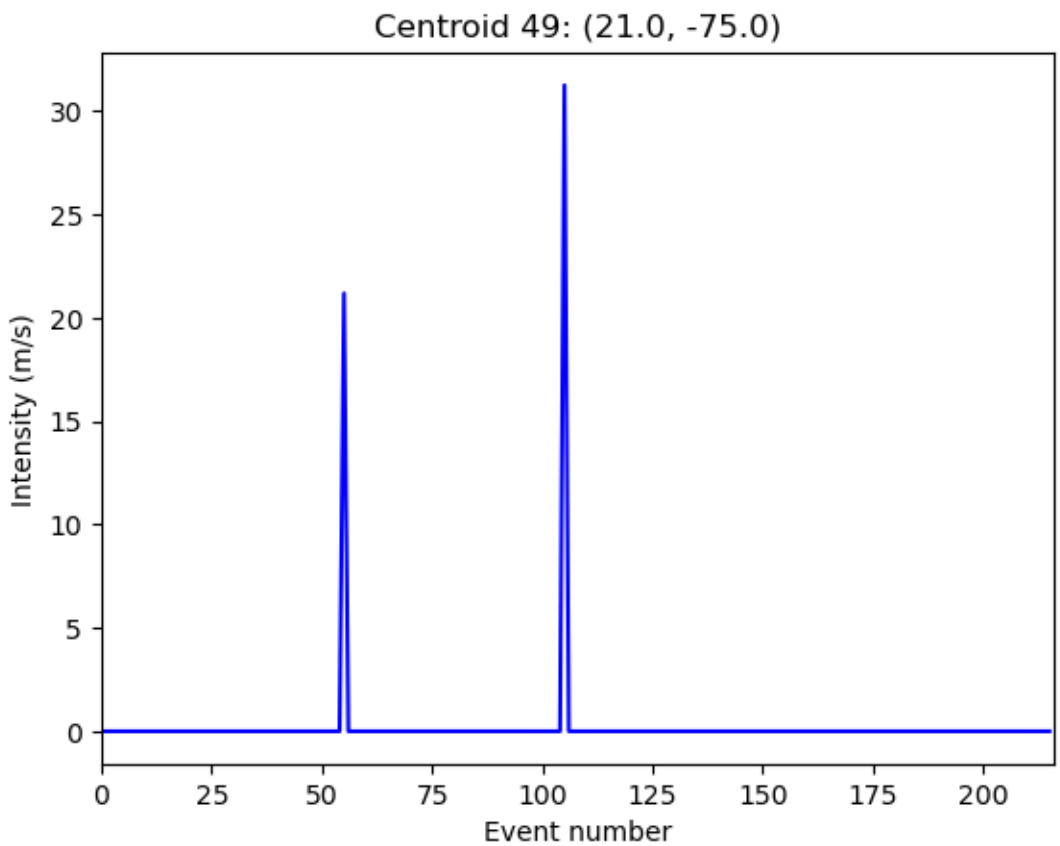



TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
Intensity (m/s)

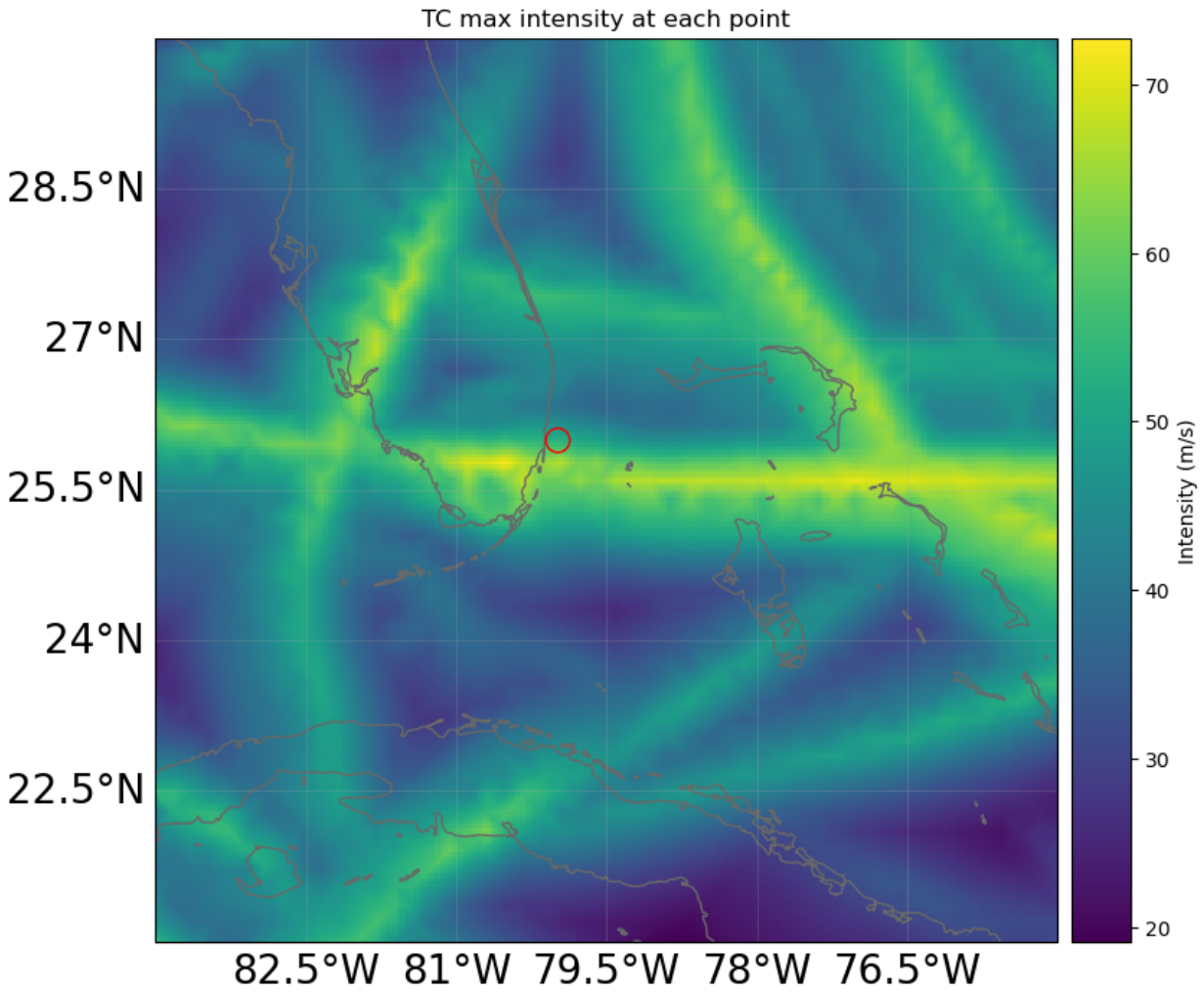

Event ID 1251: 1998295N12284
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
40
30
20
10
0
Intensity (m/s)

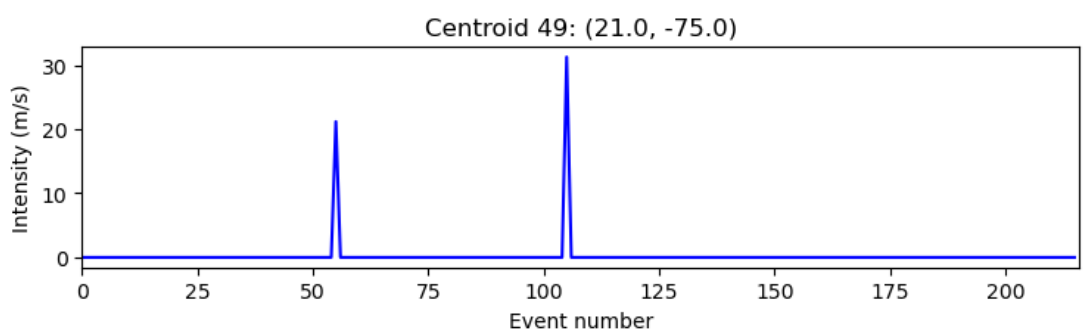

Return Period: 10 years
Return Period: 50 years
Return Period: 75 years
Return Period: 100 years
Threshold Intensity: 30 m/s
Threshold Intensity: 40 m/s
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W 81°W 79.5°W 78°W 76.5°W
Intensity (m/s)
0 10 20 30 40 50 60 70
Return Periods (years)
40 60 80 100 120 140 160 180


Centroid 50: (21.184, -84.0)
Intensity (m/s)
0
10
20
30
40
0
25
50
75
100
125
150
175
200
Event number




```
# 7. one figure with two plots: maximum intensities and selected centroid with all␣
↪intensities:
from climada.util.plot import make_map
import matplotlib.pyplot as plt

fig, ax1, fontsize = make_map(1)  # map
ax2 = fig.add_subplot(2, 1, 2)  # add regular axes
haz_tc_fl.plot_intensity(axis=ax1, event=0)  # plot original resolution
ax1.plot(-80, 26, "or", mfc="none", markersize=12)
haz_tc_fl.plot_intensity(axis=ax2, centr=(26, -80))
fig.subplots_adjust(hspace=6.5)
```


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
Intensity (m/s)


Centroid 49: (21.0, -75.0)
Intensity (m/s)
30
20
10
0
0
25
50
75
100
125
150
175
200
Event number

Part 6: Write (=save) hazards

Hazards can be written and read in hdf5 format as follows:

```
# If you see an error message, try to create a depository named results in the␣
↪repository tutorial.
haz_tc_fl.write_hdf5("results/haz_tc_fl.h5")

haz = Hazard.from_hdf5("results/haz_tc_fl.h5")
haz.check()
```

```
2024-10-01 15:53:40,418 - climada.hazard.io - INFO - Writing results/haz_tc_fl.h5
2024-10-01 15:53:40,443 - climada.hazard.centroids.centr - INFO - Writing results/haz_
↪tc_fl.h5
2024-10-01 15:53:40,461 - climada.hazard.io - INFO - Reading results/haz_tc_fl.h5
```

GeoTiff data is generated using `write_raster()`:

```
haz_ven.write_raster("results/haz_ven.tif")  # each event is a band of the tif file
```

```
2024-10-01 15:53:40,758 - climada.util.coordinates - INFO - Writting results/haz_ven.
↪tif
```

Pickle will work as well, but note that pickle has a *transient format* and should be avoided when possible:

```
from climada.util.save import save

# this generates a results folder in the current path and stores the output there
save("tutorial_haz_tc_fl.p", haz_tc_fl)
```

```
2024-10-01 15:53:40,772 - climada.util.save - INFO - Written file /Users/vgebhart/
↪Documents/climada/outputs/temp/tutorial_haz_tc_fl.p
```

2.3.2 Hazard: Tropical cyclones

Tropical cyclones tracks are gathered in the class `TCTracks` and then provided to the hazard `TropCyclone` which computes the wind gusts at each centroid. `TropCyclone` inherits from `Hazard` and has an associated hazard type `TC`.

What do tropical cyclones look like in CLIMADA?

`TCTracks` reads and handles historical tropical cyclone tracks of the IBTrACS repository or synthetic tropical cyclone tracks simulated using fully statistical or coupled statistical-dynamical modeling approaches. It also generates synthetic tracks from the historical ones using Wiener processes.

The tracks are stored in the attribute `data`, which is a list of `xarray`'s `Dataset` (see xarray.Dataset). Each `Dataset` contains the following variables:

Coordinates
time
latitude
longitude

Descriptive variables
time_step
radius_max_wind
max_sustained_wind
central_pressure
environmental_pressure

Attributes
max_sustained_wind_unit
central_pressure_unit
sid
name
orig_event_flag
data_provider
basin
id_no
category

Part 1: Load TC tracks

Records of historical TCs are very limited and therefore the database to study this natural hazard remains sparse. Only a small fraction of the TCs make landfall every year and reliable documentation of past TC landfalling events has just started in the 1950s (1980s - satellite era). The generation of synthetic storm tracks is an important tool to overcome this spatial and temporal limitation. Synthetic dataset are much larger and thus allow to estimate the risk of much rarer events. Here we show the most prominent tools in CLIMADA to load TC tracks from historical records *a)*, generate a probabilistic dataset thereof *b)*, and work with model simulations *c)*.

a) Load TC tracks from historical records

The best-track historical data from the International Best Track Archive for Climate Stewardship (IBTrACS) can easily be loaded into CLIMADA to study the historical records of TC events. The constructor `from_ibtracs_netcdf()` generates the `Datasets` for tracks selected by IBTrACS id, or by basin and year range. To achieve this, it downloads the first time the IBTrACS data v4 in netcdf format and stores it in `~/climada/data/`. The tracks can be accessed later either using the attribute `data` or using `get_track()`, which allows to select tracks by its name or id. Use the method `append()` to extend the `data` list.

If you get an error downloading the IBTrACS data, try to manually access https://www.ncei.noaa.gov/data/international-best-track-archive-for-climate-stewardship-ibtracs/v04r01/access/netcdf/, click on the file `IBTrACS.ALL.v04r01.nc` and copy it to `~/climada/data/`.

To visualize the tracks use `plot()`.

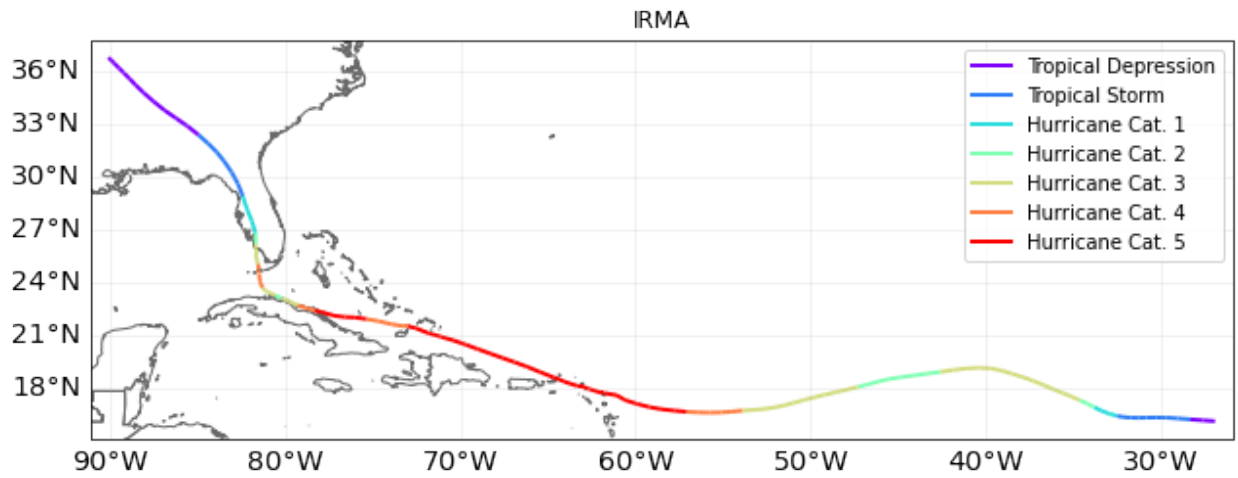
```
%matplotlib inline
from climada.hazard import TCTracks
```

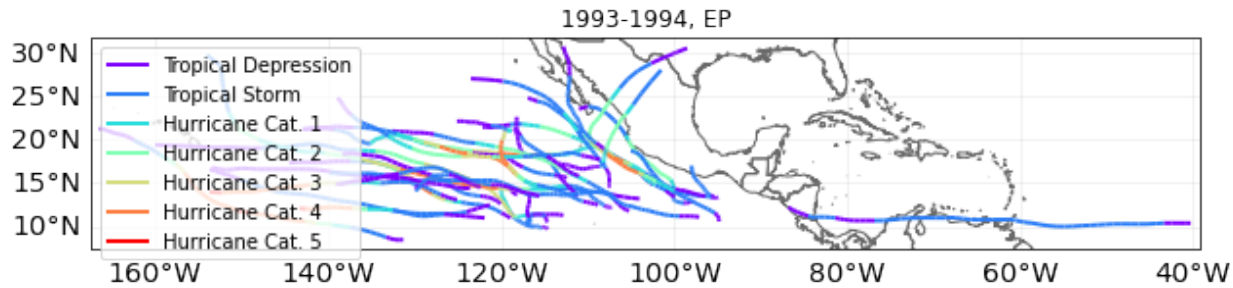
(continues on next page)

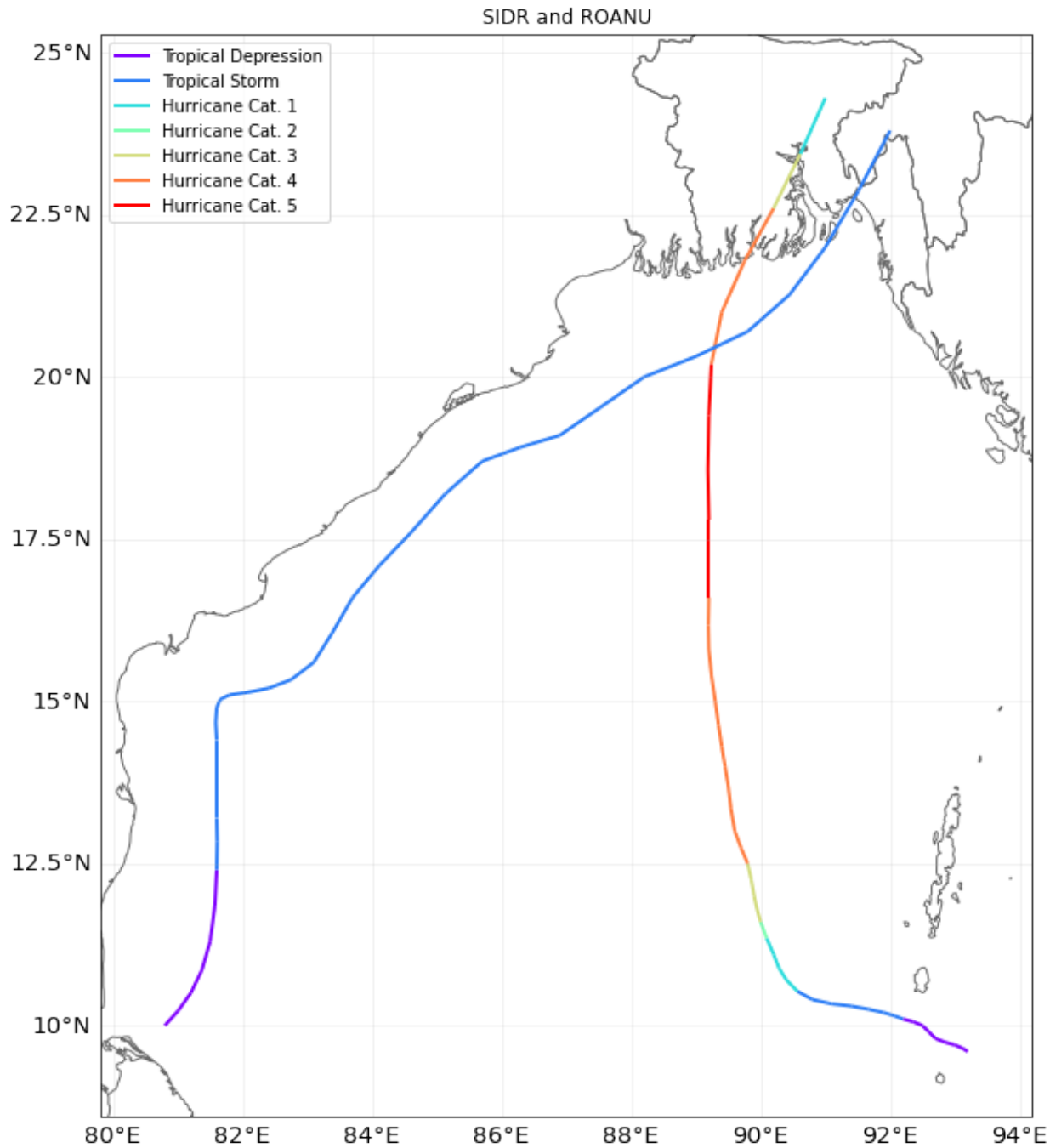
(continued from previous page)

```
tr_irma = TCTracks.from_ibtracs_netcdf(
    provider="usa", storm_id="2017242N16333"
)  # IRMA 2017
ax = tr_irma.plot()
ax.set_title("IRMA")  # set title

# other ibtracs selection options
from climada.hazard import TCTracks

# years 1993 and 1994 in basin EP.
# correct_pres ignores tracks with not enough data. For statistics (frequency of␣
↪events), these should be considered as well
sel_ibtracs = TCTracks.from_ibtracs_netcdf(
    provider="usa", year_range=(1993, 1994), basin="EP", correct_pres=False
)
print("Number of tracks:", sel_ibtracs.size)
ax = sel_ibtracs.plot()
ax.get_legend()._loc = 2  # correct legend location
ax.set_title("1993-1994, EP")  # set title

track1 = TCTracks.from_ibtracs_netcdf(
    provider="usa", storm_id="2007314N10093"
)  # SIDR 2007
track2 = TCTracks.from_ibtracs_netcdf(
    provider="usa", storm_id="2016138N10081"
)  # ROANU 2016
track1.append(track2.data)  # put both tracks together
ax = track1.plot()
ax.get_legend()._loc = 2  # correct legend location
ax.set_title("SIDR and ROANU");  # set title
```

```
2021-06-04 17:07:33,515 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-06-04 17:07:35,833 - climada.hazard.tc_tracks - WARNING - 19 storm events are␣
↪discarded because no valid wind/pressure values have been found: 1993178N14265,␣
↪1993221N12216, 1993223N07185, 1993246N16129, 1993263N11168, ...
2021-06-04 17:07:35,940 - climada.hazard.tc_tracks - INFO - Progress: 11%
2021-06-04 17:07:36,028 - climada.hazard.tc_tracks - INFO - Progress: 23%
2021-06-04 17:07:36,119 - climada.hazard.tc_tracks - INFO - Progress: 35%
2021-06-04 17:07:36,218 - climada.hazard.tc_tracks - INFO - Progress: 47%
2021-06-04 17:07:36,312 - climada.hazard.tc_tracks - INFO - Progress: 58%
2021-06-04 17:07:36,399 - climada.hazard.tc_tracks - INFO - Progress: 70%
2021-06-04 17:07:36,493 - climada.hazard.tc_tracks - INFO - Progress: 82%
2021-06-04 17:07:36,585 - climada.hazard.tc_tracks - INFO - Progress: 94%
2021-06-04 17:07:36,612 - climada.hazard.tc_tracks - INFO - Progress: 100%
Number of tracks: 33
2021-06-04 17:07:38,825 - climada.hazard.tc_tracks - INFO - Progress: 100%
2021-06-04 17:07:39,974 - climada.hazard.tc_tracks - INFO - Progress: 100%
```


IRMA
36°N
33°N
30°N
27°N
24°N
21°N
18°N
90°W
80°W
70°W
60°W
50°W
40°W
30°W
Tropical Depression
Tropical Storm
Hurricane Cat. 1
Hurricane Cat. 2
Hurricane Cat. 3
Hurricane Cat. 4
Hurricane Cat. 5


1993-1994, EP
30°N
25°N
20°N
15°N
10°N
160°W
140°W
120°W
100°W
80°W
60°W
40°W
Tropical Depression
Tropical Storm
Hurricane Cat. 1
Hurricane Cat. 2
Hurricane Cat. 3
Hurricane Cat. 4
Hurricane Cat. 5




```
tr_irma.get_track("2017242N16333")
```

```
<xarray.Dataset>
Dimensions:                 (time: 123)
Coordinates:
  * time                    (time) datetime64[ns] 2017-08-30 ... 2017-09-13T1...
    lat                     (time) float32 16.1 16.15 16.2 ... 36.2 36.5 36.8
    lon                     (time) float32 -26.9 -27.59 -28.3 ... -89.79 -90.1
Data variables:
```

(continues on next page)

(continued from previous page)

```
    time_step                (time) float64 3.0 3.0 3.0 3.0 ... 3.0 3.0 3.0 3.0
    radius_max_wind          (time) float32 60.0 60.0 60.0 ... 60.0 60.0 60.0
    radius_oci               (time) float32 180.0 180.0 180.0 ... 350.0 350.0
    max_sustained_wind       (time) float32 30.0 32.0 35.0 ... 15.0 15.0 15.0
    central_pressure         (time) float32 1.008e+03 1.007e+03 ... 1.005e+03
    environmental_pressure   (time) float64 1.012e+03 1.012e+03 ... 1.008e+03
    basin                    (time) <U2 'NA' 'NA' 'NA' 'NA' ... 'NA' 'NA' 'NA'
Attributes:
    max_sustained_wind_unit:  kn
    central_pressure_unit:    mb
    name:                     IRMA
    sid:                      2017242N16333
    orig_event_flag:          True
    data_provider:            ibtracs_usa
    id_no:                    2017242016333.0
    category:                 5
```

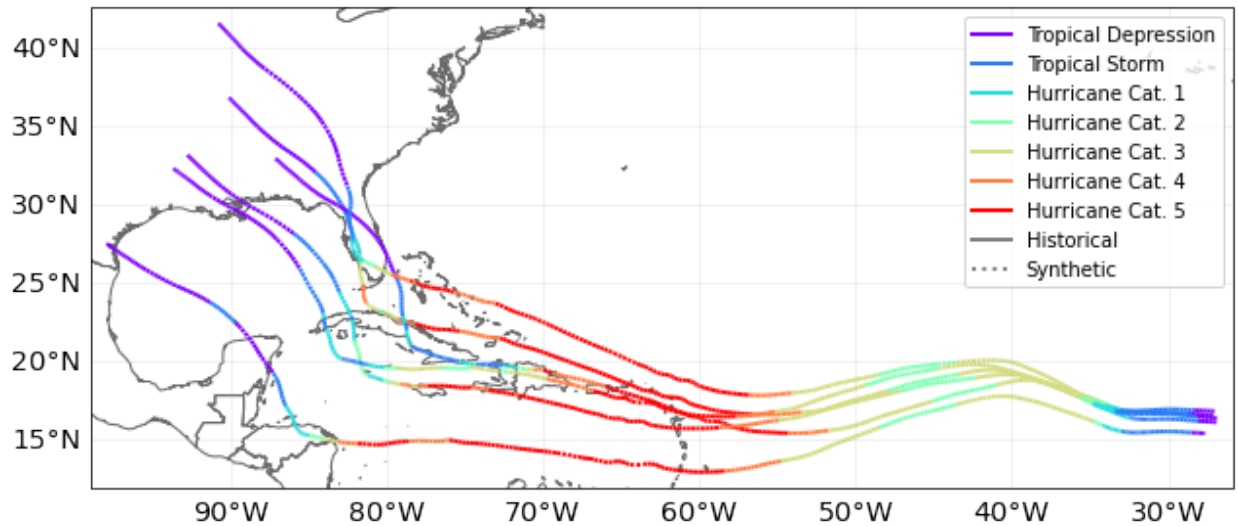
b) Generate probabilistic events

Once tracks are present in `TCTracks`, one can generate synthetic tracks for each present track based on directed random walk. Note that the tracks should be interpolated to use the same timestep **before** generation of probabilistic events.

`calc_perturbed_trajectories()` generates an ensemble of "nb_synth_tracks" numbers of synthetic tracks is computed for every track. The methodology perturbs the tracks locations, and if decay is `True` it additionally includes decay of wind speed and central pressure drop after landfall. No other track parameter is perturbed.

```
# here we use tr_irma retrieved from IBTrACS with the function above
# select number of synthetic tracks (nb_synth_tracks) to generate per present tracks.
tr_irma.equal_timestep()
tr_irma.calc_perturbed_trajectories(nb_synth_tracks=5)
tr_irma.plot();
# see more configutration options (e.g. amplitude of max random starting point shift␣
↪in decimal degree; max_shift_ini)
```

```
<GeoAxesSubplot:>
```




```
tr_irma.data[-1]  # last synthetic track. notice the value of orig_event_flag and name
```

```
<xarray.Dataset>
Dimensions:                  (time: 349)
Coordinates:
  * time                     (time) datetime64[ns] 2017-08-30 ... 2017-09-13T1...
    lon                      (time) float64 -27.64 -27.8 -27.96 ... -97.81 -97.93
    lat                      (time) float64 15.39 15.41 15.42 ... 27.41 27.49
Data variables:
    time_step                (time) float64 1.0 1.0 1.0 1.0 ... 1.0 1.0 1.0 1.0
    radius_max_wind          (time) float64 60.0 60.0 60.0 ... 60.0 60.0 60.0
    radius_oci               (time) float64 180.0 180.0 180.0 ... 350.0 350.0
    max_sustained_wind       (time) float64 30.0 30.67 31.33 ... 15.0 14.99 14.96
    central_pressure         (time) float64 1.008e+03 1.008e+03 ... 1.005e+03
    environmental_pressure   (time) float64 1.012e+03 1.012e+03 ... 1.008e+03
    basin                    (time) <U2 'NA' 'NA' 'NA' 'NA' ... 'NA' 'NA' 'NA'
    on_land                  (time) bool False False False ... False True True
    dist_since_lf            (time) float64 nan nan nan nan ... nan 7.605 22.71
Attributes:
    max_sustained_wind_unit:  kn
    central_pressure_unit:    mb
    name:                     IRMA_gen5
    sid:                      2017242N16333_gen5
    orig_event_flag:          False
    data_provider:            ibtracs_usa
    id_no:                    2017242016333.05
    category:                 5
```

EXERCISE

Using the first synthetic track generated,

1. Which is the time frequency of the data?
2. Compute the maximum sustained wind for each day.

```
# Put your code here
```

```
# SOLUTION:
import numpy as np

# select the track
tc_syn = tr_irma.get_track("2017242N16333_gen1")

# 1. Which is the time frequency of the data?
# The values of a DataArray are numpy.arrays.
# The nummpy.ediff1d computes the different between elements in an array
diff_time_ns = np.ediff1d(tc_syn["time"])
diff_time_h = diff_time_ns.astype(int) / 1000 / 1000 / 1000 / 60 / 60
print("Mean time frequency in hours:", diff_time_h.mean())
print("Std time frequency in hours:", diff_time_h.std())
print()

# 2. Compute the maximum sustained wind for each day.
print(
    "Daily max sustained wind:", tc_syn["max_sustained_wind"].groupby("time.day").
↪max()
)
```

```
Mean time frequency in hours: 1.0
Std time frequency in hours: 0.0

Daily max sustained wind: <xarray.DataArray 'max_sustained_wind' (day: 15)>
array([100.        , 100.        , 100.        , 123.33333333,
       155.        , 155.        , 150.        , 138.        ,
        51.85384486,  58.03963987,  29.03963987,   3.57342356,
         3.35512013,  54.        ,  99.        ])
Coordinates:
  * day      (day) int64 1 2 3 4 5 6 7 8 9 10 11 12 13 30 31
```

c) ECMWF Forecast Tracks

ECMWF publishes tropical cyclone forecast tracks free of charge as part of the WMO essentials. These tracks are detected automatically in the ENS and HRES models. The non-supervised nature of the model may lead to artefacts.

The `tc_fcast` trackset below inherits from TCTracks, but contains some additional metadata that follows ECMWF's definitions. Try plotting these tracks and compare them to the official cones of uncertainty! The example track at `tc_fcast.data[0]` shows the data structure.

```
# This functionality is part of climada_petals, uncomment to execute
# from climada_petals.hazard import TCForecast
#
# tc_fcast = TCForecast()
# tc_fcast.fetch_ecmwf()
#
# print(tc_fcast.data[0])
```

d) Load TC tracks from other sources

In addition to the *historical records of TCs (IBTrACS)*, the *probabilistic extension* of these tracks, and the *ECMWF Forecast tracks*, CLIMADA also features functions to read in synthetic TC tracks from other sources. These include synthetic storm tracks from Kerry Emanuel's coupled statistical-dynamical model (Emanuel et al., 2006 as used in Geiger et al., 2016), from an open source derivative of Kerry Emanuel's model FAST, synthetic storm tracks from a second coupled statistical-dynamical model (CHAZ) (as described in Lee et al., 2018), and synthetic storm tracks from a fully statistical model (STORM) Bloemendaal et al., 2020). However, these functions are partly under development and/or targeted at advanced users of CLIMADA in the context of very specific use cases. They are thus not covered in this tutorial.

Part 2: TropCyclone() class

The `TropCyclone` class is a derived class of *Hazard*. As such, it contains all the attributes and methods of a `Hazard`. Additionally, it comes with the constructor method `from_tracks` to model tropical cyclones from tracks contained in a `TCTracks` instance.

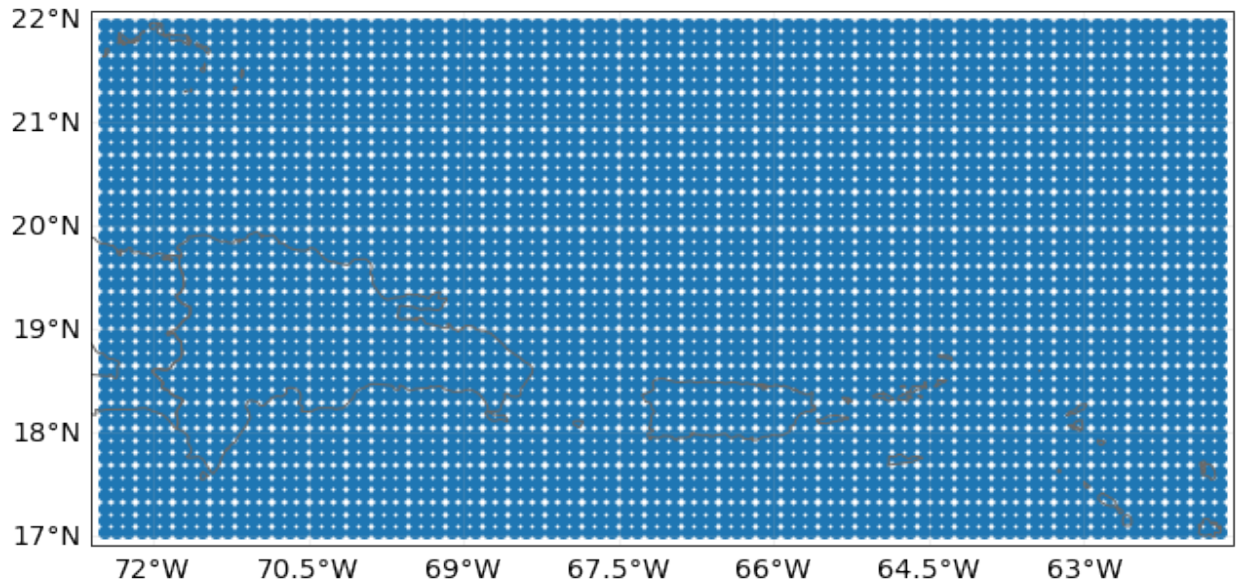
When setting tropical cyclones from tracks, the centroids where to map the wind gusts (the hazard intensity) can be provided. If no centroids are provided, the global centroids `GLB_NatID_grid_0360as_adv_2.mat` are used.

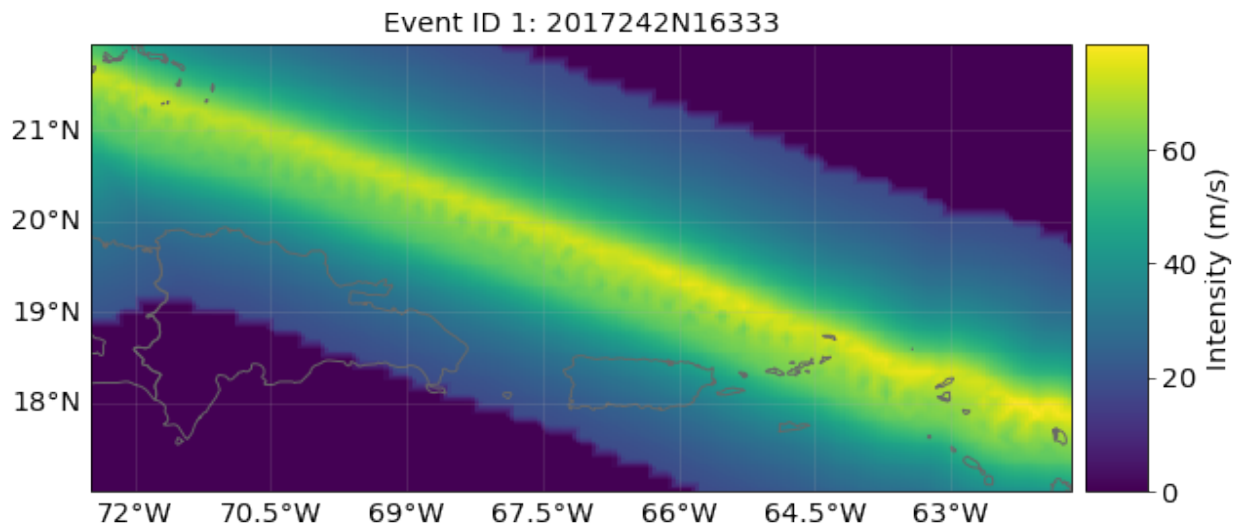
From the track properties the 1 min sustained peak gusts are computed in each centroid as the sum of a circular wind field (following Holland, 2008) and the translational wind speed that arises from the storm movement. We incorporate the decline of the translational component from the cyclone centre by multiplying it by an attenuation factor. See CLIMADA v1 and references therein for more information.

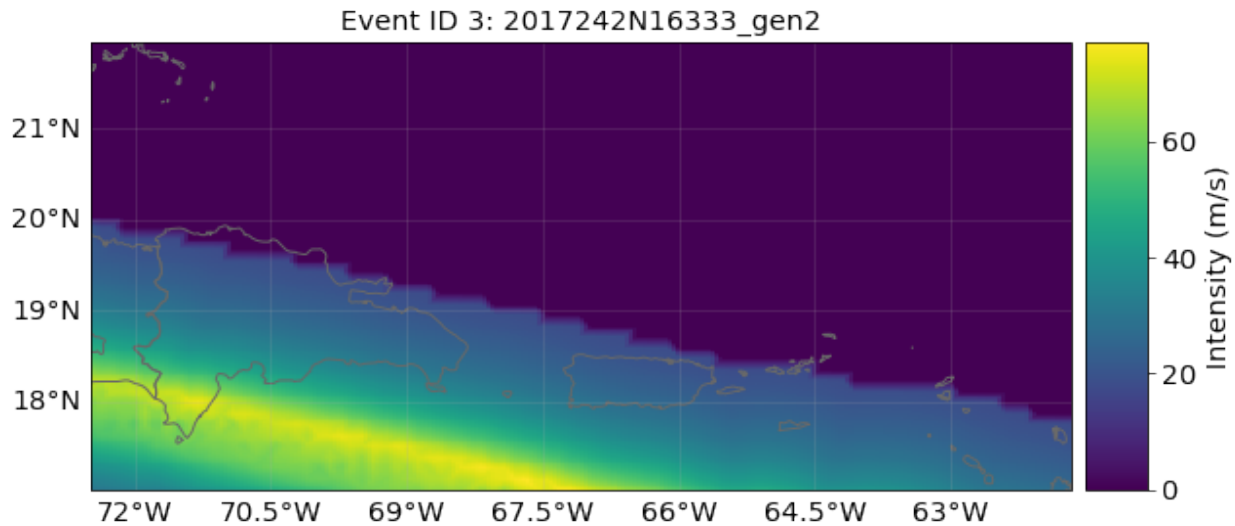
a) Default hazard generation for tropical cyclones

```
from climada.hazard import Centroids, TropCyclone

# construct centroids
min_lat, max_lat, min_lon, max_lon = 16.99375, 21.95625, -72.48125, -61.66875
cent = Centroids.from_pnt_bounds((min_lon, min_lat, max_lon, max_lat), res=0.12)
cent.plot()

# construct tropical cyclones
tc_irma = TropCyclone.from_tracks(tr_irma, centroids=cent)
# tc_irma = TropCyclone.from_tracks(tr_irma)  # try without given centroids. It might␣
↪take too much space of your memory
# and then the kernel will be killed: So, don't use this function without given␣
↪centroids!
tc_irma.check()
tc_irma.plot_intensity("2017242N16333")
# IRMA
tc_irma.plot_intensity("2017242N16333_gen2");  # IRMA's synthetic track 2
```

```
<GeoAxesSubplot:title={'center':'Event ID 3: 2017242N16333_gen2'}>
```





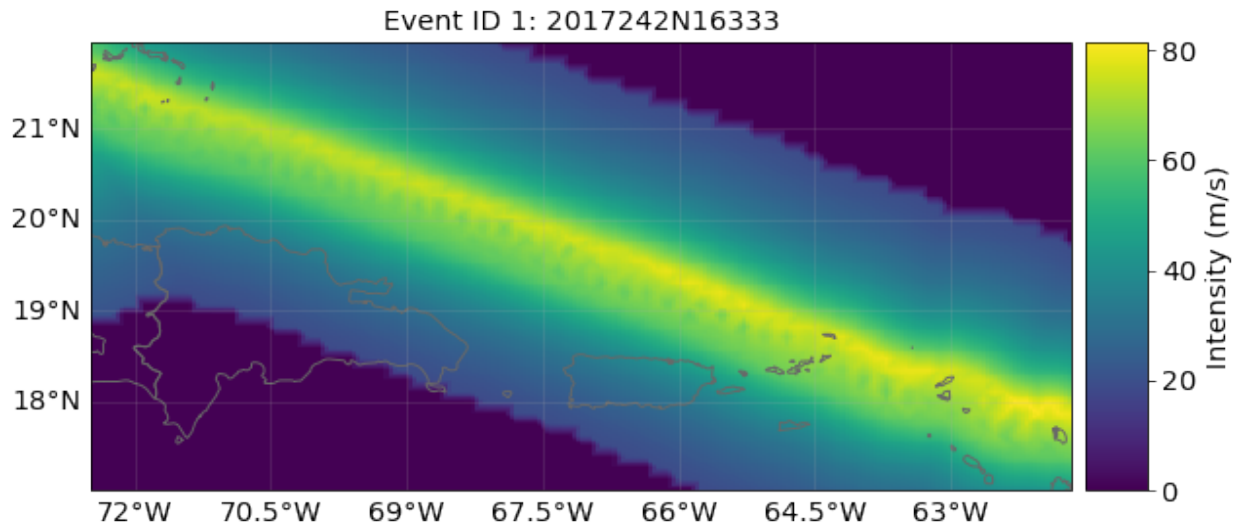


b) Implementing climate change

`apply_climate_scenario_knu` implements the changes in frequency due to climate change described in Knutson et al (2020) and Jewson et al. (2021). This requires to pass the rcp scenario of interest, the projection's future reference year, the projection's percentile of interest and the historical baseline period. For simplicity we keep these latter two as default values only spefify the rcp (45) and the future reference year (2055).

```
# an Irma event-like in 2055 under RCP 4.5:
tc_irma = TropCyclone.from_tracks(tr_irma, centroids=cent)
tc_irma_cc = tc_irma.apply_climate_scenario_knu(target_year=2055, scenario="4.5")

rel_freq_incr = np.round(
    (np.mean(tc_irma_cc.frequency) - np.mean(tc_irma.frequency))
    / np.mean(tc_irma.frequency)
    * 100,
    0,
)

print(
    f"\nA TC like Irma would undergo a frequency increase of about {rel_freq_incr} %␣
↪in 2055 under RCP 45"
)
```

```
<GeoAxesSubplot:title={'center':'Event ID 1: 2017242N16333'}>
```



Note: this method to implement climate change is simplified and does only take into account changes in TC frequency. However, how hurricane damage changes with climate remains challenging to assess. Records of hurricane damage exhibit widely fluctuating values because they depend on rare, landfalling events which are substantially more volatile than the underlying basin-wide TC characteristics. For more accurate future projections of how a warming climate might shape TC characteristics, there is a two-step process needed. First, the understanding of how climate change affects critical environmental factors (like SST, humidity, etc.) that shape TCs is required. Second, the means of simulating how these changes impact TC characteristics (such as intensity, frequency, etc.) are necessary. Statistical-dynamical models (Emanuel et al., 2006 and Lee et al., 2018) are physics-based and allow for such climate change studies. However, this goes beyond the scope of this tutorial.

c) Multiprocessing - improving performance for big computations

Multiprocessing is part of the tropical cyclone module. Simply provide a process pool as method argument. Below is an example of how large amounts of data could be processed.

WARNING: Running multiprocessing code from Jupyter Notebooks can be cumbersome. It's suggested to copy the code and paste it into an interactive python console.

```
from climada.hazard import TCTracks, Centroids, TropCyclone

from pathos.pools import ProcessPool as Pool
pool = Pool() # start a pathos pool

lon_min, lat_min, lon_max, lat_max = -160, 10, -100, 36
centr = Centroids.from_pnt_bounds((lon_min, lat_min, lon_max, lat_max), 0.1)

tc_track = TCTracks.from_ibtracs_netcdf(provider='usa', year_range=(1992, 1994),␣
↪basin='EP')
tc_track.equal_timestep(pool=pool)
tc_track.calc_perturbed_trajectories(pool=pool) # OPTIONAL: if you want to generate a␣
↪probabilistic set of TC tracks.

tc_haz = TropCyclone.from_tracks(tc_track, centroids=centr, pool=pool)
tc_haz.check()
```

(continues on next page)

(continued from previous page)

```
pool.close()
pool.join()
```

d) Making videos

Videos of a tropical cyclone hitting specific centroids can be created with the method `video_intensity()`.

WARNING: Creating an animated gif file may consume a lot of memory, up to the point where the os starts swapping or even an 'out-of-memory' exception is thrown.

```
# Note: execution of this cell will fail unless there is enough memory available (>␣
↪10G)

from climada.hazard import Centroids, TropCyclone, TCTracks

track_name = "2017242N16333"  #'2016273N13300' #'1992230N11325'

tr_irma = TCTracks.from_ibtracs_netcdf(provider="usa", storm_id="2017242N16333")

lon_min, lat_min, lon_max, lat_max = -83.5, 24.4, -79.8, 29.6
centr_video = Centroids.from_pnt_bounds((lon_min, lat_min, lon_max, lat_max), 0.04)
centr_video.check()

tc_video = TropCyclone()

tc_list, tr_coord = tc_video.video_intensity(
    track_name, tr_irma, centr_video, file_name="results/irma_tc_fl.gif"
)
```

```
2022-04-08 10:01:29,114 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2022-04-08 10:01:31,696 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617␣
↪(1/1)
2022-04-08 10:01:38,120 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪11374 coastal centroids.
2022-04-08 10:01:38,135 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,144 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12032 coastal centroids.
2022-04-08 10:01:38,158 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,170 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,184 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,194 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,207 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,217 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,232 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,241 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
```

(continues on next page)

(continued from previous page)

```
2022-04-08 10:01:38,256 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,267 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,285 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,296 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,313 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,323 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,338 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,348 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,364 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,374 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,391 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,400 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,416 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,427 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,441 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,452 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:01:38,470 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:01:38,479 - climada.hazard.trop_cyclone - INFO - Generating video irma_
↪tc_fl.gif
```

```
15it [00:53,  3.60s/it]
```

`tc_list` contains a list with TropCyclone instances plotted at each time step `tr_coord` contains a list with the track path coordinates plotted at each time step

Saving disk space with mp4

Animated gif images occupy a lot of space. Using mp4 as output format makes the video sequences much smaller! However this requires the package *ffmpeg* to be installed, which is not part of the ordinary climada environment. It can be installed by executing the following command in a console:

```
conda install ffmpeg
```

Creating the same videa as above in mp4 format can be done in this way then:

```
# Note: execution of this cell will fail unless there is enough memory available (>␣
↪12G) and ffmpeg is installed

import shutil
from matplotlib import animation
from matplotlib.pyplot import rcParams

rcParams["animation.ffmpeg_path"] = shutil.which("ffmpeg")
writer = animation.FFMpegWriter(bitrate=500)
```

(continues on next page)

(continued from previous page)

```
tc_list, tr_coord = tc_video.video_intensity(
    track_name, tr_irma, centr_video, file_name="results/irma_tc_fl.mp4",␣
↪writer=writer
)
```

```
2022-04-08 10:03:27,161 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪11374 coastal centroids.
2022-04-08 10:03:27,182 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,192 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12032 coastal centroids.
2022-04-08 10:03:27,207 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,218 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,235 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,247 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,263 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,275 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,294 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,304 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,319 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,333 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,350 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,363 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,382 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,393 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,412 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,422 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,442 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,453 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,471 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,480 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,496 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,507 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,523 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,533 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪12314 coastal centroids.
2022-04-08 10:03:27,548 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2022-04-08 10:03:27,559 - climada.hazard.trop_cyclone - INFO - Generating video irma_
↪tc_fl.mp4
```

```
15it [00:55,  3.71s/it]
```

REFERENCES:

- Bloemendaal, N., Haigh, I. D., de Moel, H., Muis, S., Haarsma, R. J., & Aerts, J. C. J. H. (2020). Generation of a global synthetic tropical cyclone hazard dataset using STORM. Scientific Data, 7(1). https://doi.org/10.1038/s41597-020-0381-2
- Emanuel, K., S. Ravela, E. Vivant, and C. Risi, 2006: A Statistical Deterministic Approach to Hurricane Risk Assessment. Bull. Amer. Meteor. Soc., 87, 299–314, https://doi.org/10.1175/BAMS-87-3-299.
- Geiger, T., Frieler, K., & Levermann, A. (2016). High-income does not protect against hurricane losses. Environmental Research Letters, 11(8). https://doi.org/10.1088/1748-9326/11/8/084012
- Knutson, T. R., Sirutis, J. J., Zhao, M., Tuleya, R. E., Bender, M., Vecchi, G. A., … Chavas, D. (2015). Global projections of intense tropical cyclone activity for the late twenty-first century from dynamical downscaling of CMIP5/RCP4.5 scenarios. Journal of Climate, 28(18), 7203–7224. https://doi.org/10.1175/JCLI-D-15-0129.1
- Lee, C. Y., Tippett, M. K., Sobel, A. H., & Camargo, S. J. (2018). An environmentally forced tropical cyclone hazard model. Journal of Advances in Modeling Earth Systems, 10(1), 223–241. https://doi.org/10.1002/2017MS001186

2.3.3 Hazard: winter windstorms / extratropical cyclones in Europe

Or: The StormEurope hazard subclass of CLIMADA

Auth: Jan Hartman & Thomas Röösli
Date: 2018-04-26 & 2020-03-03

This notebook will give a quick tour of the capabilities of the StormEurope hazard class. This includes functionalities to apply probabilistic alterations to historical storms.

```
%matplotlib inline
import matplotlib.pyplot as plt

plt.rcParams["figure.figsize"] = [15, 10]
```

Reading Data

StormEurope was written under the presumption that you'd start out with WISC storm footprint data in netCDF format. This notebook works with a demo dataset. If you would like to work with the real data: (1) Please follow the link and download the file C3S_WISC_FOOTPRINT_NETCDF_0100.tgz from the Copernicus Windstorm Information Service, (2) unzip it (3) uncomment the last two lines in the following codeblock and (4) adjust the variable "WISC_files".

We first construct an instance and then point the reader at a directory containing compatible `.nc` files. Since there are other files in there, we must be explicit and use a globbing pattern; supplying incompatible files will make the reader fail.

The reader actually calls `climada.util.files_handler.get_file_names`, so it's also possible to hand it an explicit list of filenames, or a dirname, or even a list of glob patterns or directories.

```
from climada.hazard import StormEurope
from climada.util.constants import WS_DEMO_NC

storm_instance = StormEurope.from_footprints(WS_DEMO_NC)

# WISC_files = '/path/to/folder/C3S_WISC_FOOTPRINT_NETCDF_0100/fp_era[!er5]*_0.nc'
# storm_instance = StormEurope.from_footprints(WISC_files)
```

Introspection

Let's quickly see what attributes this class brings with it:

```
?storm_instance
```

```
Type:           StormEurope
String form:    <climada.hazard.storm_europe.StormEurope object at 0x7f2a986b4c70>
File:           ~/code/climada_python/climada/hazard/storm_europe.py
Docstring:
A hazard set containing european winter storm events. Historic storm
events can be downloaded at https://cds.climate.copernicus.eu/ and read
with `from_footprints`. Weather forecasts can be automatically downloaded from
https://opendata.dwd.de/ and read with from_icon_grib(). Weather forecast
from the COSMO-Consortium https://www.cosmo-model.org/ can be read with
from_cosmoe_file().

Attributes
----------
ssi_wisc : np.array, float
    Storm Severity Index (SSI) as recorded in
    the footprint files; apparently not reproducible from the footprint
    values only.
ssi : np.array, float
    SSI as set by set_ssi; uses the Dawkins
    definition by default.
Init docstring: Calls the Hazard init dunder. Sets unit to 'm/s'.
```

You could also try listing all permissible methods with `dir(storm_instance)`, but since that would include the methods from the `Hazard` base class, you wouldn't know what's special. The best way is to read the source: uncomment the following statement to read more.

```
# StormEurope??
```

Into the Storm Severity Index (SSI)

The SSI, according to Dawkins et al. 2016 or Lamb and Frydendahl, 1991, can be set using `set_ssi`. For demonstration purposes, I show the default arguments. (Check also the defaults using `storm_instance.calc_ssi?`, the method for which set_ssi is a wrapper.)

We won't be using the `plot_ssi` functionality just yet, because we only have two events; the graph really isn't informative. After this, we'll generate some more storms to make that plot more aesthetically pleasing.

```
storm_instance.set_ssi(
    method="wind_gust",
    intensity=storm_instance.intensity,
    # the above is just a more explicit way of passing the default
    on_land=True,
    threshold=25,
    sel_cen=None,
    # None is default. sel_cen could be used to subset centroids
)
```

Probabilistic Storms

This class allows generating probabilistic storms from historical ones according to a method outlined in Schwierz et al. 2010. This means that per historical event, we generate 29 new ones with altered intensities. Since it's just a bunch of vector operations, this is pretty fast.

However, we should not return the entire probabilistic dataset in-memory: in trials, this used up 60 GB of RAM, thus requiring a great amount of swap space. Instead, we must select a country by setting the `reg_id` parameter to an ISO_N3 country code used in the Natural Earth dataset. It is also possible to supply a list of ISO codes. If your machine is up for the job of handling the whole dataset, set the `reg_id` parameter to `None`.

Since assigning each centroid a country ID is a rather inefficient affair, you may need to wait a minute or two for the entire WISC dataset to be processed. For the small demo dataset, it runs pretty quickly.

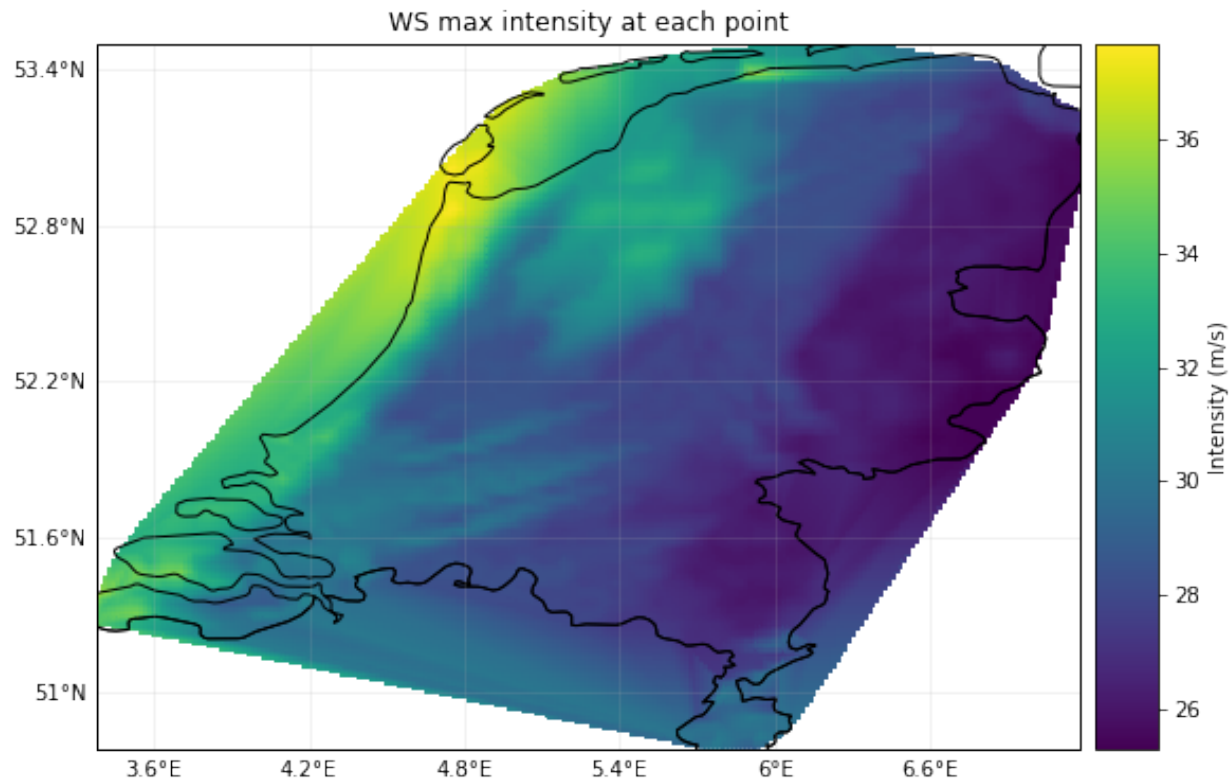
```
%%time
storm_prob = storm_instance.generate_prob_storms(reg_id=528)
storm_prob.plot_intensity(0);
```

```
2020-03-05 10:29:31,845 - climada.hazard.centroids.centr - INFO - Setting geometry␣
↪points.
2020-03-05 10:29:32,248 - climada.hazard.centroids.centr - DEBUG - Setting region_id␣
↪9944 points.
2020-03-05 10:29:32,466 - climada.util.coordinates - DEBUG - Setting region_id 9944␣
↪points.
2020-03-05 10:29:33,506 - climada.hazard.storm_europe - INFO - Commencing␣
↪probabilistic calculations
2020-03-05 10:29:33,620 - climada.hazard.storm_europe - INFO - Generating new␣
↪StormEurope instance
2020-03-05 10:29:33,663 - climada.util.checker - DEBUG - Hazard.ssi not set.
2020-03-05 10:29:33,664 - climada.util.checker - DEBUG - Hazard.ssi_wisc not set.
2020-03-05 10:29:33,665 - climada.util.checker - DEBUG - Hazard.event_name not set.␣
↪Default values set.
```

```
C:\shortpaths\GitHub\climada_python\climada\util\plot.py:311: UserWarning: Tight␣
↪layout not applied. The left and right margins cannot be made large enough to␣
↪accommodate all axes decorations.
  fig.tight_layout()
```

```
Wall time: 2.24 s
```

```
<cartopy.mpl.geoaxes.GeoAxesSubplot at 0x1dafba69940>
```

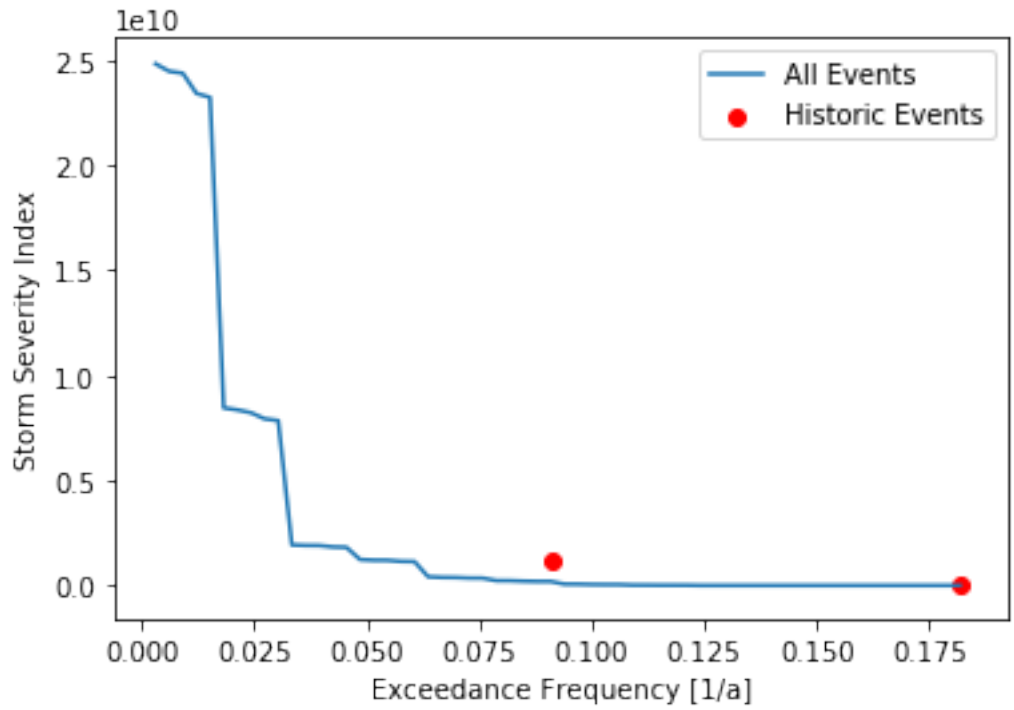


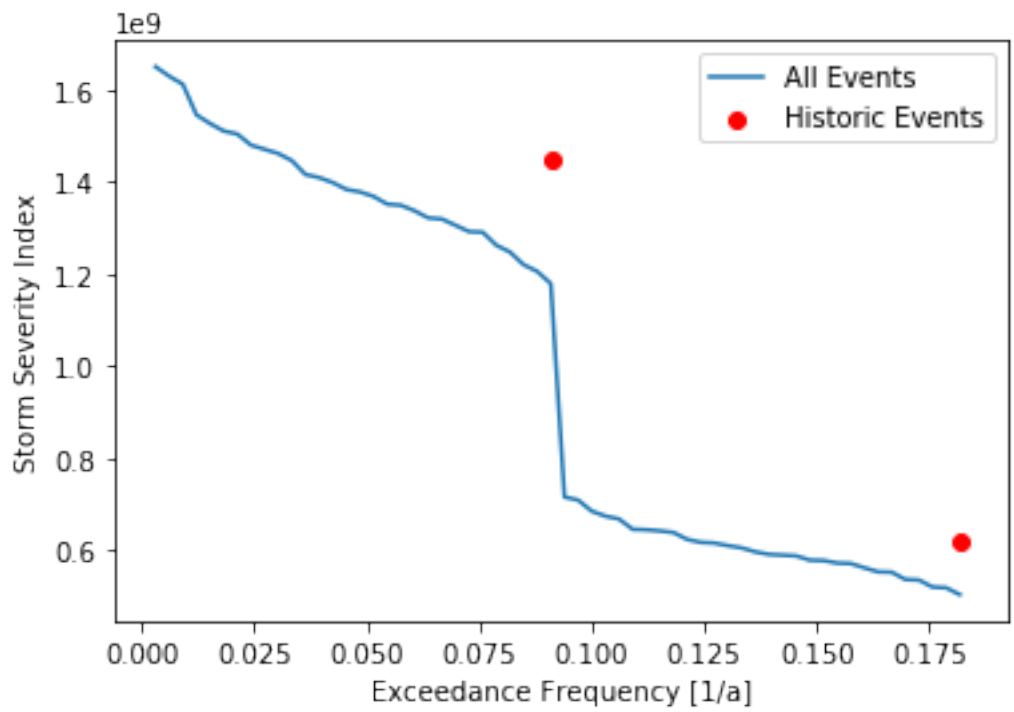
We can get much more fancy in our calls to `generate_prob_storms`; the keyword arguments after `ssi_args` are passed on to `_hist2prob`, allowing us to tweak the probabilistic permutations.

```
ssi_args = {
    "on_land": True,
    "threshold": 25,
}

storm_prob_xtreme = storm_instance.generate_prob_storms(
    reg_id=[56, 528],  # BEL and NLD
    spatial_shift=2,
    ssi_args=ssi_args,
    power=1.5,
    scale=0.3,
)
```

We can now check out the SSI plots of both these calculations. The comparison between the historic and probabilistic ssi values, only makes sense for the full dataset.

```
storm_prob_xtreme.plot_ssi(full_area=True)
storm_prob.plot_ssi(full_area=True);
```






```
(<Figure size 1080x720 with 1 Axes>,
 <AxesSubplot:xlabel='Exceedance Frequency [1/a]', ylabel='Storm Severity Index'>)
```

2.3.4 Using the Copernicus Seasonal Forecast Tools package to create a hazard object

Introduction

The copernicus-seasonal-forecast-tools package was developed to manage seasonal forecast data from the Copernicus Climate Data Store (CDS) for the U-CLIMADAPT project. It offers comprehensive tools for downloading, processing, computing climate indices, and generating hazard objects based on seasonal forecast datasets, particularly Seasonal forecast daily and subdaily data on single levels. The package is tailored to integrate seamlessly with the CLIMADA, supporting climate risk assessment and the development of effective adaptation strategies.

Features:

- Automated download of the high-dimensional seasonal forecasts data via the Copernicus API
- Preprocessing of sub-daily forecast data into daily formats
- Calculation of heat-related climate indices (e.g., heatwave days, tropical nights)
- Conversion of processed indices into CLIMADA hazard objects ready for impact modelling
- Flexible modular architecture to accommodate additional indices or updates to datasets

In this tutorial, you can see a simple example of how to retrieve and process data from Copernicus, calculate a heat-related index, and create a hazard object. For more detailed documentation and advanced examples, please visit the repository or the documentation.

Prerequisites:

1. CDS account and API key: Register at https://cds.climate.copernicus.eu
2. CDS API client installation: pip install cdsapi
3. CDS API configuration: Create a .cdsapirc file in your home directory with your API key and URL. For instructions, visit: https://cds.climate.copernicus.eu/how-to-api#install-the-cds-api-client
4. Dataset Terms and Conditions: After selecting the dataset to download, make sure to accept the terms and conditions on the corresponding dataset webpage in the CDS portal before running this notebook. Here, https://cds.climate.copernicus.eu/datasets/seasonal-original-single-levels?tab=download.

For more information, visit the comprehensive CDS API setup guide, which walks you through each step of the process. Once configured, you'll be ready to explore and analyze seasonal forecast data.

Note: Ensure you have the **necessary permissions** and comply with the CDS data usage policies when using this package. You can view the terms and conditions at https://cds.climate.copernicus.eu/datasets/seasonal-original-single-levels?tab=download. You can find them at the bottom of the download page.

```
# Import packages

import warnings
import datetime as dt

warnings.filterwarnings("ignore")
from seasonal_forecast_tools import SeasonalForecast, ClimateIndex
from seasonal_forecast_tools.utils.coordinates_utils import bounding_box_from_
↪countries
from seasonal_forecast_tools.utils.time_utils import month_name_to_number
```

Set up parameters

To configure the package for working with Copernicus forecast data and converting it into a hazard object for CLIMADA, you will need to define several essential parameters. These settings are crucial as they specify the type of data to be retrieved, the format, the forecast period, and the geographical area of interest. These parameters influence how the forecast data is processed and transformed into a hazard object.

Below, we outline these parameters and use an example for the `Tmax` – Maximum Temperature index to demonstrate the seasonal forecast functionality.

To learn more about what these parameters entail and their significance, please refer to the documentation on the CDS webpage.

Overview of parameters

index_metric: Defines the type of index to be calculated. There are currently **12 predefined options** available, including temperature-based indices (`Tmean` – Mean Temperature, `Tmin` – Minimum Temperature, `Tmax` – Maximum Temperature), heat stress indicators (`HIA` – Heat Index Adjusted, `HIS` – Heat Index Simplified, `HUM` – Humidex, `AT` – Apparent Temperature, `WBGT` – Wet Bulb Globe Temperature (Simple)), and extreme event indices (`HW` – Heat Wave, `TR` – Tropical Nights, `TX30` – Hot Days).

- **Heat Waves ("HW")**:
  If `index_metric` is set to 'HW' for heat wave calculations, additional parameters can be specified to fine-tune the heat wave detection:
  - **threshold**: Temperature threshold above which days are considered part of a heat wave. Default is 27°C.
  - **min_duration**: Minimum number of consecutive days above the threshold required to define a heat wave event. Default is 3 days.
  - **max_gap**: Maximum allowable gap (in days) between two heat wave events to consider them as one single event. Default is 0 days.
- **Tropical Nights ("TR")**:
  If `index_metric` is set to 'TR' for tropical nights, an additional parameter can be specified to set the threshold:
  - **threshold**: Nighttime temperature threshold, above which a night is considered "tropical." Default is 20°C.
- ⚠ **Flexibility:** Users can define and integrate their own indices into the pipeline to extend the analysis according to their specific needs.

format : Specifies the format of the data to be downloaded, "grib" or "netcdf". Copernicus do **NOT** recommended netcdf format for operational workflows since conversion to netcdf is considered experimental. More information here.

originating_centre: Identifies the source of the data. A standard choice is "dwd" (German Weather Service), one of eight providers including ECMWF, UK Met Office, Météo France, CMCC, NCEP, JMA, and ECCC.

system: Refers to a specific model or configuration used for forecasts. In this script, the default value is "21," which corresponds to the GCSF (German Climate Forecast System) version 2.1. More details can be found in the CDS documentation.

year_list: A list of years for which data should be downloaded and processed.

initiation_month: A list of the months in which the forecasts are initiated. Example: ["March", "April"].

forecast_period: Specifies the months relative to the forecast's initiation month for which the data is forecasted. Example: ["June", "July", "August"] indicates forecasts for these months. The maximum available is 7 months.

- ⍰ **Important**: When an initiation month is in one year and the forecast period in the next, the system recognizes the forecast extends beyond the initial year. Data is retrieved based on the initiation month, with lead times covering the following year. The forecast is stored under the initiation year's directory, ensuring consistency while spanning both years.

area_selection: This determines the geographical area for which the data should be downloaded. It can be set to

- Global coverage:
  - Use the predefined function bounding_box_global() to select the entire globe.
- Custom geographical bounds (cardinal coordinates):
  - Input explicit latitude/longitude limits (in EPSG:4326).
  - *bounds = bounding_box_from_cardinal_bounds(northern=49, eastern=20, southern=40, western=10)*
- Country codes (ISO alpha-3):
  - Provide a list of ISO 3166-1 alpha-3 country codes (e.g., "DEU" for Germany, "CHE" for Switzerland). The bounding box is constructed as the union of all selected countries.See this wikipedia page for the country codes.
  - *bounds = bounding_box_from_countries(["CHE", "DEU"])*

overwrite: Boolean flag that, when set to True, forces the system to redownload and reprocess existing files.

```
# We define above parameters for an example on Tmax
index_metric = ClimateIndex.Tmax.name
data_format = "grib"  # 'grib' or 'netcdf'
originating_centre = "dwd"
system = "21"
forecast_period = [
    "December",
    "February",
]  # from December to February including January
year_list = [2022]
initiation_month = ["November"]
overwrite = False
bounds = bounding_box_from_countries(["URY"])

# Parameters for Heat Waves
hw_threshold = 27
hw_min_duration = 3
hw_max_gap = 0

# Parameters for Tropical Nights
threshold_tr = 20

# Describe the selected climate index and the associated input data
forecast = SeasonalForecast(
    index_metric=index_metric,
    year_list=year_list,
    forecast_period=forecast_period,
    initiation_month=initiation_month,
    bounds=bounds,
    data_format=data_format,
    originating_centre=originating_centre,
    system=system,
)
```

The variables required for your selected index will be printed below. This allows you to see which data will be accessed and helps estimate the data volume.

```
forecast.explain_index()
```

```
'Explanation for Maximum Temperature: Maximum Temperature: Tracks the highest␣
↪temperature recorded over a specified period. Required variables: 2m_temperature'
```

Download and Process Data

You can now call the `forecast.download_and_process_data` method, which efficiently retrieves and organizes Copernicus forecast data. It checks for existing files to avoid redundant downloads, stores data by format (grib or netCDF), year, month. Then the files are processed for further analysis, such as calculating climate indices or creating hazard objects within CLIMADA. Here are the aspects of this process:

- **Data Download**: The method downloads the forecast data for the selected years, months, and regions. The data is retrieved in **grib** or **netCDF** formats, which are commonly used for storing meteorological data. If the required files already exist in the specified directories, the system will skip downloading them, as indicated by the log messages such as:
  "Corresponding grib file SYSTEM_DIR/copernicus_data/seasonal_forecasts/dwd/sys21/2023/init03/valid06_08/downloaded_data/grib
  already exists."
- **Data Processing**: After downloading (or confirming the existence of) the files, the system converts them into daily **netCDF** files. Each file contains gridded, multi-ensemble data for daily mean, maximum, and minimum, structured by forecast step, ensemble member, latitude, and longitude. The log messages confirm the existence or creation of these files, for example:
  "Daily file SYSTEM_DIR/copernicus_data/seasonal_forecasts/dwd/sys21/2023/init03/valid06_08/processed_data/TX30_boundsW4_S
  already exists."
- **Geographic and Temporal Focus**: The files are generated for a specific time frame (e.g., June and July 2022) and a predefined geographic region, as specified by the parameters such as `bounds`, `month_list`, and `year_list`. This ensures that only the selected data for your analysis is downloaded and processed.
- **Data Completeness**: Messages like "already exists" ensure that you do not redundantly download or process data, saving time and computing resources. However, if the data files are missing, they will be downloaded and processed as necessary.

```
# Download and process data
forecast.download_and_process_data()
```

```
{'downloaded_data': {'2022_init11_valid12_02': PosixPath('/Users/daraya/climada/data/
↪copernicus_data/seasonal_forecasts/dwd/sys21/2022/init11/valid12_02/downloaded_data/
↪grib/Tmax_boundsN-59_S-35_E-52_W-29.grib')},
 'processed_data': {'2022_init11_valid12_02': PosixPath('/Users/daraya/climada/data/
↪copernicus_data/seasonal_forecasts/dwd/sys21/2022/init11/valid12_02/processed_data/
↪Tmax_boundsN-59_S-35_E-52_W-29.nc')}}
```

From here, you can consult the data created by calling xarray. This will display the structure of the dataset, including dimensions such as time (here called steps), latitude, longitude, and ensemble members, as well as coordinates, data variables such as the processed daily values of temperature at two meters (mean, max, and min), and associated metadata and attributes.

This already processed daily data can be used as needed; or you can now also calculate a heat-related index as in the following cells.

```
import xarray as xr
```

(continues on next page)

(continued from previous page)

```
file_path = "/Users/daraya/climada/data/copernicus_data/seasonal_forecasts/dwd/sys21/
↪2022/init11/valid12_02/processed_data/Tmax_boundsN-59_S-35_E-52_W-29.nc"
ds = xr.open_dataset(file_path)
ds
```

```
<xarray.Dataset> Size: 3MB
Dimensions:     (number: 50, step: 90, latitude: 7, longitude: 8)
Coordinates:
  * number      (number) int64 400B 0 1 2 3 4 5 6 7 ... 42 43 44 45 46 47 48 49
    time        datetime64[ns] 8B ...
  * step        (step) timedelta64[ns] 720B 30 days 09:00:00 ... 119 days 09:...
    surface     float64 8B ...
  * latitude    (latitude) float64 56B -29.95 -30.95 -31.95 ... -34.95 -35.95
  * longitude   (longitude) float64 64B -59.43 -58.43 -57.43 ... -53.43 -52.43
    valid_time  (step) datetime64[ns] 720B ...
Data variables:
    t2m_mean    (number, step, latitude, longitude) float32 1MB ...
    t2m_max     (number, step, latitude, longitude) float32 1MB ...
    t2m_min     (number, step, latitude, longitude) float32 1MB ...
```

```
# You can also just select the data for the first time step of the first ensemble␣
↪member
ds.isel(step=0, number=0)
```

```
<xarray.Dataset> Size: 832B
Dimensions:     (latitude: 7, longitude: 8)
Coordinates:
    number      int64 8B 0
    time        datetime64[ns] 8B ...
    step        timedelta64[ns] 8B 30 days 09:00:00
    surface     float64 8B ...
  * latitude    (latitude) float64 56B -29.95 -30.95 -31.95 ... -34.95 -35.95
  * longitude   (longitude) float64 64B -59.43 -58.43 -57.43 ... -53.43 -52.43
    valid_time  datetime64[ns] 8B ...
Data variables:
    t2m_mean    (latitude, longitude) float32 224B ...
    t2m_max     (latitude, longitude) float32 224B ...
    t2m_min     (latitude, longitude) float32 224B ...
```

Calculate Climate Indices

If you decide to calculate an index, you can call the `forecast.calculate_index` method to compute specific climate indices (such as Maximum Temperature). The output is automatically saved and organized in a structured format for further analysis. Here are some details:

- **Index Calculation**: The method processes seasonal forecast data to compute the selected index for the chosen years, months, and regions. This index represents a specific climate condition, such as the number of Maximum Temperature ("Tmax") over the forecast period, as defined in the parameters.
- **Data Storage**: The calculated index data is saved in **netCDF** format. These files are automatically saved in directories specific to the index and time period. The file paths are printed below the processing steps. For example, the computed index values are stored in:

"SYSTEM_DIR/copernicus_data/seasonal_forecasts/dwd/sys21/2023/init03/valid06_08/indices/TX30/TX30_boundsW4_S44_E11_N

Similarly, the statistics of the index (e.g., mean, max, min, std) are saved in:
"SYSTEM_DIR/copernicus_data/seasonal_forecasts/dwd/sys21/2023/init03/valid06_08/indices/TX30/TX30_boundsW4_S44_E11_N
These files ensure that both the raw indices and their statistical summaries are available for detailed analysis.

Each file contains data for a specific month and geographic region, as defined in the parameters. This allows you to analyze how the selected climate index varies over time and across different locations.

- **Completeness of Data Processing**: Messages 'Index Tmax successfully calculated and saved for…' confirm the successful calculation and storage of the index, ensuring that all requested data has been processed and saved correctly.

```
# Calculate index
forecast.calculate_index(
    hw_threshold=hw_threshold, hw_min_duration=hw_min_duration, hw_max_gap=hw_max_gap
)
```

```
{'2022_init11_valid12_02': {'daily': PosixPath('/Users/daraya/climada/data/copernicus_
↪data/seasonal_forecasts/dwd/sys21/2022/init11/valid12_02/indices/Tmax/Tmax_boundsN-
↪59_S-35_E-52_W-29_daily.nc'),
  'monthly': PosixPath('/Users/daraya/climada/data/copernicus_data/seasonal_forecasts/
↪dwd/sys21/2022/init11/valid12_02/indices/Tmax/Tmax_boundsN-59_S-35_E-52_W-29_
↪monthly.nc'),
  'stats': PosixPath('/Users/daraya/climada/data/copernicus_data/seasonal_forecasts/
↪dwd/sys21/2022/init11/valid12_02/indices/Tmax/Tmax_boundsN-59_S-35_E-52_W-29_stats.
↪nc')}}
```

We can explore the properties of the daily file containing the calculated index and, for example, visualize the values for each ensemble member on a specific date. This enables a quick visual inspection of how the predicted Tmax varies across ensemble members on that day.

```
# Call the daily index file
ds_daily = xr.open_dataset(
    "/Users/daraya/climada/data/copernicus_data/seasonal_forecasts/dwd/sys21/2022/
↪init11/valid12_02/indices/Tmax/Tmax_boundsN-59_S-35_E-52_W-29_daily.nc"
)
ds_daily
```

```
<xarray.Dataset> Size: 1MB
Dimensions:    (number: 50, latitude: 7, longitude: 8, step: 90)
Coordinates:
  * number     (number) int64 400B 0 1 2 3 4 5 6 7 8 ... 42 43 44 45 46 47 48 49
    time       datetime64[ns] 8B ...
    surface    float64 8B ...
  * latitude   (latitude) float64 56B -29.95 -30.95 -31.95 ... -34.95 -35.95
  * longitude  (longitude) float64 64B -59.43 -58.43 -57.43 ... -53.43 -52.43
  * step       (step) timedelta64[ns] 720B 30 days 09:00:00 ... 119 days 09:0...
Data variables:
    Tmax       (number, step, latitude, longitude) float32 1MB ...
```

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

(continues on next page)

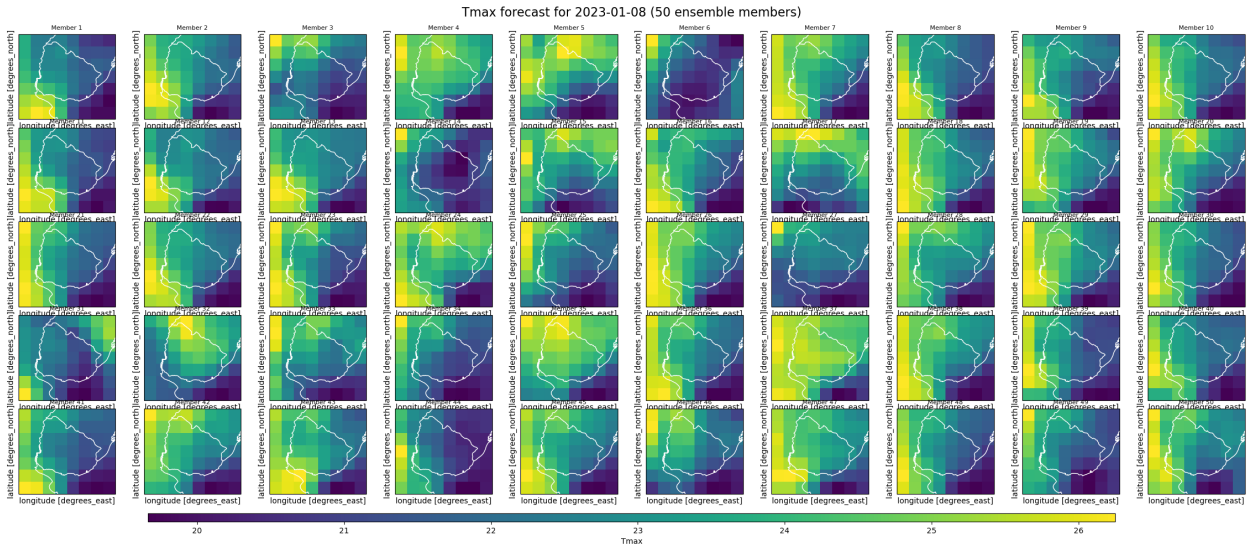
(continued from previous page)

```
import cartopy.crs as ccrs
import cartopy.feature as cfeature

# Plot all the member for one day
target_date = "2023-01-08"
index_metric = "Tmax"

init_time = pd.to_datetime(ds_daily["time"].values.item())  # scalar datetime
steps = ds_daily["step"].values  # timedelta64[ns]

forecast_dates = init_time + pd.to_timedelta(steps)  # Compute real forecast dates

target_datetime = pd.to_datetime(
    target_date
)  # Find matching step index for target date
index_match = np.where(forecast_dates.normalize() == target_datetime.normalize())[0]

if len(index_match) == 0:
    raise ValueError(
        f"Date {target_date} not found in forecast data.\nAvailable dates: {forecast_
↪dates.strftime('%Y-%m-%d').tolist()}"
    )
step_index = index_match[0]

data = ds_daily[index_metric].isel(step=step_index)

fig, axs = plt.subplots(
    5, 10, figsize=(25, 10), subplot_kw={"projection": ccrs.PlateCarree()}
)
axs = axs.flatten()

for i in range(50):
    ax = axs[i]
    p = data.isel(number=i).plot(
        ax=ax,
        transform=ccrs.PlateCarree(),
        x="longitude",
        y="latitude",
        add_colorbar=False,
        cmap="viridis",
    )
    ax.coastlines(color="white")
    ax.add_feature(cfeature.BORDERS, edgecolor="white")
    ax.set_title(f"Member {i+1}", fontsize=8)
    ax.set_xticks([])
    ax.set_yticks([])

plt.subplots_adjust(
    bottom=0.1, top=0.93, left=0.05, right=0.95, wspace=0.1, hspace=0.1
)  # Add shared colorbar
cbar_ax = fig.add_axes([0.15, 0.05, 0.7, 0.015])
fig.colorbar(p, cax=cbar_ax, orientation="horizontal", label=index_metric)
```

(continues on next page)

(continued from previous page)

```
plt.suptitle(
    f"{index_metric} forecast for {target_date} (50 ensemble members)", fontsize=16
)
plt.show()
```



We can also access the monthly index data, where the `step` coordinate now represents **monthly values** instead of daily ones, reflecting the aggregation over each forecast month.

```
import datetime

ds_monthly = xr.open_dataset(
    "/Users/daraya/climada/data/copernicus_data/seasonal_forecasts/dwd/sys21/2022/
↪init11/valid12_02/indices/Tmax/Tmax_boundsN-59_S-35_E-52_W-29_monthly.nc"
)
for step in ds_monthly.step.values:
    print(str(step))
```

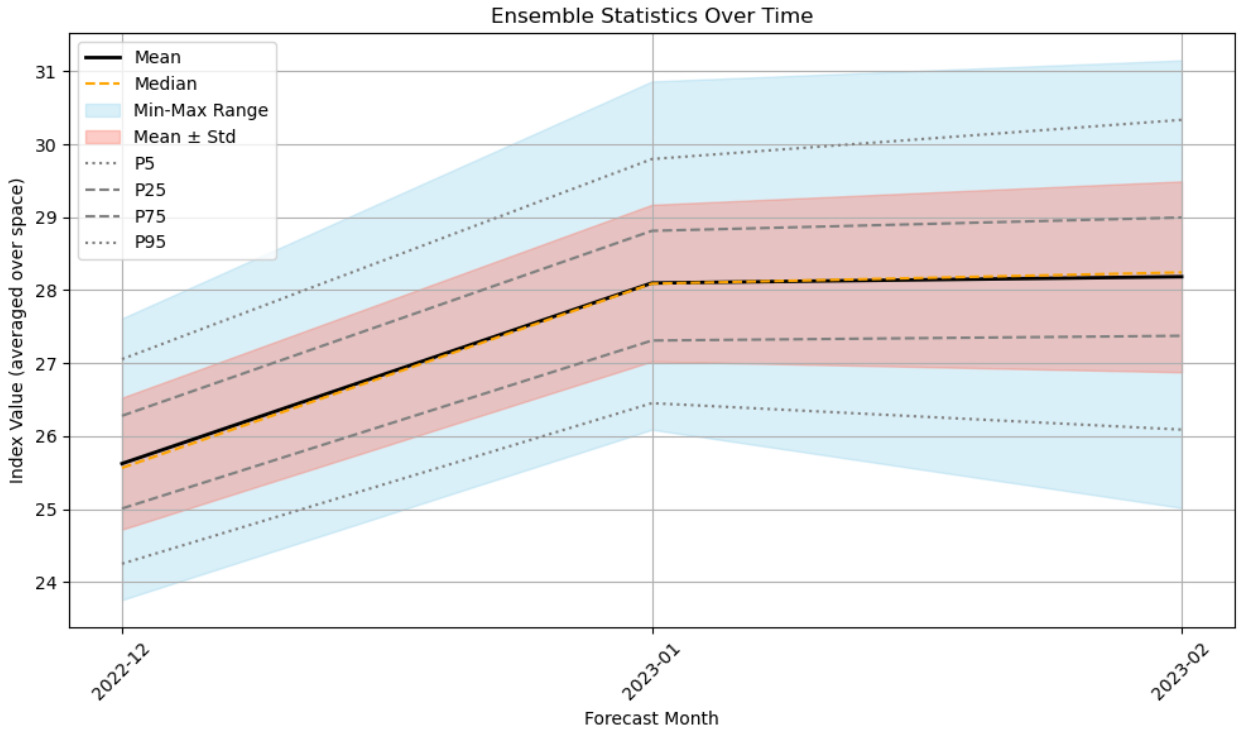
```
2022-12
2023-01
2023-02
```

You can now also explore ensemble statistics over time using the precomputed `stats.nc` file. The file contains ten statistical variables: mean, median, max, min, standard deviation, and percentiles from p5 to p95, computed across ensemble members. These stored variables allow for easy plotting and visualization of how ensemble members vary across forecast months. As expected, the spread in member predictions tends to increase toward the end of the forecast period.

```
# Call the statistics file
ds_stats = xr.open_dataset(
    "/Users/daraya/climada/data/copernicus_data/seasonal_forecasts/dwd/sys21/2022/
↪init11/valid12_02/indices/Tmax/Tmax_boundsN-59_S-35_E-52_W-29_stats.nc"
)
ds_stats
```

```
<xarray.Dataset> Size: 10kB
Dimensions:          (latitude: 7, longitude: 8, step: 3)
Coordinates:
    time             datetime64[ns] 8B ...
    surface          float64 8B ...
  * latitude         (latitude) float64 56B -29.95 -30.95 ... -34.95 -35.95
  * longitude        (longitude) float64 64B -59.43 -58.43 ... -53.43 -52.43
  * step             (step) <U7 84B '2022-12' '2023-01' '2023-02'
    quantile         float64 8B ...
Data variables:
    ensemble_mean    (step, latitude, longitude) float32 672B ...
    ensemble_median  (step, latitude, longitude) float32 672B ...
    ensemble_max     (step, latitude, longitude) float32 672B ...
    ensemble_min     (step, latitude, longitude) float32 672B ...
    ensemble_std     (step, latitude, longitude) float32 672B ...
    ensemble_p5      (step, latitude, longitude) float64 1kB ...
    ensemble_p25     (step, latitude, longitude) float64 1kB ...
    ensemble_p50     (step, latitude, longitude) float64 1kB ...
    ensemble_p75     (step, latitude, longitude) float64 1kB ...
    ensemble_p95     (step, latitude, longitude) float64 1kB ...
```

```
# Extract statistics
steps = ds_stats["step"].values
mean = ds_stats["ensemble_mean"].mean(dim=["latitude", "longitude"])
median = ds_stats["ensemble_median"].mean(dim=["latitude", "longitude"])
std = ds_stats["ensemble_std"].mean(dim=["latitude", "longitude"])
min_ = ds_stats["ensemble_min"].mean(dim=["latitude", "longitude"])
max_ = ds_stats["ensemble_max"].mean(dim=["latitude", "longitude"])
p5 = ds_stats["ensemble_p5"].mean(dim=["latitude", "longitude"])
p25 = ds_stats["ensemble_p25"].mean(dim=["latitude", "longitude"])
p75 = ds_stats["ensemble_p75"].mean(dim=["latitude", "longitude"])
p95 = ds_stats["ensemble_p95"].mean(dim=["latitude", "longitude"])

# Plot
plt.figure(figsize=(10, 6))
plt.plot(steps, mean, label="Mean", color="black", linewidth=2)
plt.plot(steps, median, label="Median", color="orange", linestyle="--")
plt.fill_between(steps, min_, max_, color="skyblue", alpha=0.3, label="Min-Max Range")
plt.fill_between(
    steps, mean - std, mean + std, color="salmon", alpha=0.4, label="Mean ± Std"
)
plt.plot(steps, p5, label="P5", linestyle=":", color="gray")
plt.plot(steps, p25, label="P25", linestyle="--", color="gray")
plt.plot(steps, p75, label="P75", linestyle="--", color="gray")
plt.plot(steps, p95, label="P95", linestyle=":", color="gray")
plt.title("Ensemble Statistics Over Time")
plt.xlabel("Forecast Month")
plt.ylabel("Index Value (averaged over space)")
plt.xticks(rotation=45)
plt.grid(True)
plt.legend()
plt.tight_layout()
plt.show()
```



Calculate a Hazard Object

Then you can call `forecast.process_and_save_hazards` method to convert processed index from Copernicus forecast data into a hazard object.

- **Hazard Object Creation**: The method processes seasonal forecast data for specified years and months, converting these into hazard objects. These objects encapsulate potential risks associated with specific weather events or conditions, such as Maximum Temperature ('Tmax') indicated in the parameters, over the forecast period.
- **Data Storage**: The hazard data for each ensemble member of the forecast is saved as HDF5 files. These files are automatically stored in specific directories corresponding to each month and type of hazard. The file paths are printed below the processing steps. For example, *"/SYSTEM_DIR/copernicus_data/seasonal_forecasts/dwd/sys21/2023/init03/valid06_08/hazard/TX30/TX30_boundsW4_S44_E11_N48.h* HDF5 is a versatile data model that efficiently stores large volumes of complex data.

Each file is specific to a particular month and hazard scenario ('Tmax' in this case) and covers all ensemble members for that forecast period, aiding in detailed risk analysis.

- **Completeness of Data Processing**: Messages like 'Completed processing for 2022-07. Data saved in…' confirm the successful processing and storage of the hazard data for that period, ensuring that all requested data has been properly handled and stored.

```
from climada.hazard import Hazard

forecast.save_index_to_hazard()
```

```
{'2022_init11_valid12_02': PosixPath('/Users/daraya/climada/data/copernicus_data/
↪seasonal_forecasts/dwd/sys21/2022/init11/valid12_02/hazard/Tmax/Tmax_boundsN-59_S-
↪35_E-52_W-29.hdf5')}
```

You can always inspect the properties of the `Hazard` object or visualize its contents. Noted the `date` attribute uses **serial**

date numbers (ordinal format), which is common in climate data. To convert these to standard datetime format, you can use datetime.datetime.fromordinal.

```
# Load the hazard and plot intensity for the selected grid
forecast.save_index_to_hazard()
initiation_month_str = f"{month_name_to_number(initiation_month[0]):02d}"
forecast_month_str = f"{forecast.valid_period_str[-2:]}"  # Last month in valid period
forecast_year = year_list[0]

path_to_hazard = forecast.get_pipeline_path(
    forecast_year, initiation_month_str, "hazard"
)
hazard = Hazard.from_hdf5(path_to_hazard)

# Access hazard attributes
print("Hazard attributes:")
print(" - Shape of intensity (time, gridpoint):", hazard.intensity.shape)
print(" - Centroids:", hazard.centroids.shape)
print(" - Units:", hazard.units)
print(" - event_id:", hazard.event_id)
print(" - frequency:", hazard.frequency)
print(" - min, max fraction:", hazard.fraction.min(), hazard.fraction.max())
print(" - Date:", hazard.date)
print("min, max fraction: ", hazard.fraction.min(), hazard.fraction.max())
print(" - event_name:"), hazard.event_name
```

```
Hazard attributes:
 - Shape of intensity (time, gridpoint): (150, 56)
 - Centroids: (7, 8)
 - Units: °C
 - event_id: [  1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18
  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36
  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54
  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72
  73  74  75  76  77  78  79  80  81  82  83  84  85  86  87  88  89  90
  91  92  93  94  95  96  97  98  99 100 101 102 103 104 105 106 107 108
 109 110 111 112 113 114 115 116 117 118 119 120 121 122 123 124 125 126
 127 128 129 130 131 132 133 134 135 136 137 138 139 140 141 142 143 144
 145 146 147 148 149 150]
 - frequency: [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.
 1. 1. 1. 1. 1. 1.]
 - min, max fraction: 0.0 0.0
 - Date: [738490 738521 738552 738490 738521 738552 738490 738521 738552 738490
 738521 738552 738490 738521 738552 738490 738521 738552 738490 738521
 738552 738490 738521 738552 738490 738521 738552 738490 738521 738552
 738490 738521 738552 738490 738521 738552 738490 738521 738552 738490
 738521 738552 738490 738521 738552 738490 738521 738552 738490 738521
 738552 738490 738521 738552 738490 738521 738552 738490 738521 738552
```

(continues on next page)

(continued from previous page)

```
 738490 738521 738552 738490 738521 738552 738490 738521 738552 738490
 738521 738552 738490 738521 738552 738490 738521 738552 738490 738521
 738552 738490 738521 738552 738490 738521 738552 738490 738521 738552
 738490 738521 738552 738490 738521 738552 738490 738521 738552 738490
 738521 738552 738490 738521 738552 738490 738521 738552 738490 738521
 738552 738490 738521 738552 738490 738521 738552 738490 738521 738552
 738490 738521 738552 738490 738521 738552 738490 738521 738552 738490
 738521 738552 738490 738521 738552 738490 738521 738552 738490 738521
 738552 738490 738521 738552 738490 738521 738552 738490 738521 738552]
min, max fraction:  0.0 0.0
 - event_name:
```

```
(None,
 ['member0',
  'member0',
  'member0',
  'member1',
  'member1',
  'member1',
  'member2',
  'member2',
  'member2',
  'member3',
  'member3',
  'member3',
  'member4',
  'member4',
  'member4',
  'member5',
  'member5',
  'member5',
  'member6',
  'member6',
  'member6',
  'member7',
  'member7',
  'member7',
  'member8',
  'member8',
  'member8',
  'member9',
  'member9',
  'member9',
  'member10',
  'member10',
  'member10',
  'member11',
  'member11',
  'member11',
  'member12',
  'member12',
  'member12',
```

(continues on next page)

(continued from previous page)

```
 'member13',
 'member13',
 'member13',
 'member14',
 'member14',
 'member14',
 'member15',
 'member15',
 'member15',
 'member16',
 'member16',
 'member16',
 'member17',
 'member17',
 'member17',
 'member18',
 'member18',
 'member18',
 'member19',
 'member19',
 'member19',
 'member20',
 'member20',
 'member20',
 'member21',
 'member21',
 'member21',
 'member22',
 'member22',
 'member22',
 'member23',
 'member23',
 'member23',
 'member24',
 'member24',
 'member24',
 'member25',
 'member25',
 'member25',
 'member26',
 'member26',
 'member26',
 'member27',
 'member27',
 'member27',
 'member28',
 'member28',
 'member28',
 'member29',
 'member29',
 'member29',
 'member30',
```

(continues on next page)

(continued from previous page)

```
 'member30',
 'member30',
 'member31',
 'member31',
 'member31',
 'member32',
 'member32',
 'member32',
 'member33',
 'member33',
 'member33',
 'member34',
 'member34',
 'member34',
 'member35',
 'member35',
 'member35',
 'member36',
 'member36',
 'member36',
 'member37',
 'member37',
 'member37',
 'member38',
 'member38',
 'member38',
 'member39',
 'member39',
 'member39',
 'member40',
 'member40',
 'member40',
 'member41',
 'member41',
 'member41',
 'member42',
 'member42',
 'member42',
 'member43',
 'member43',
 'member43',
 'member44',
 'member44',
 'member44',
 'member45',
 'member45',
 'member45',
 'member46',
 'member46',
 'member46',
 'member47',
 'member47',
```

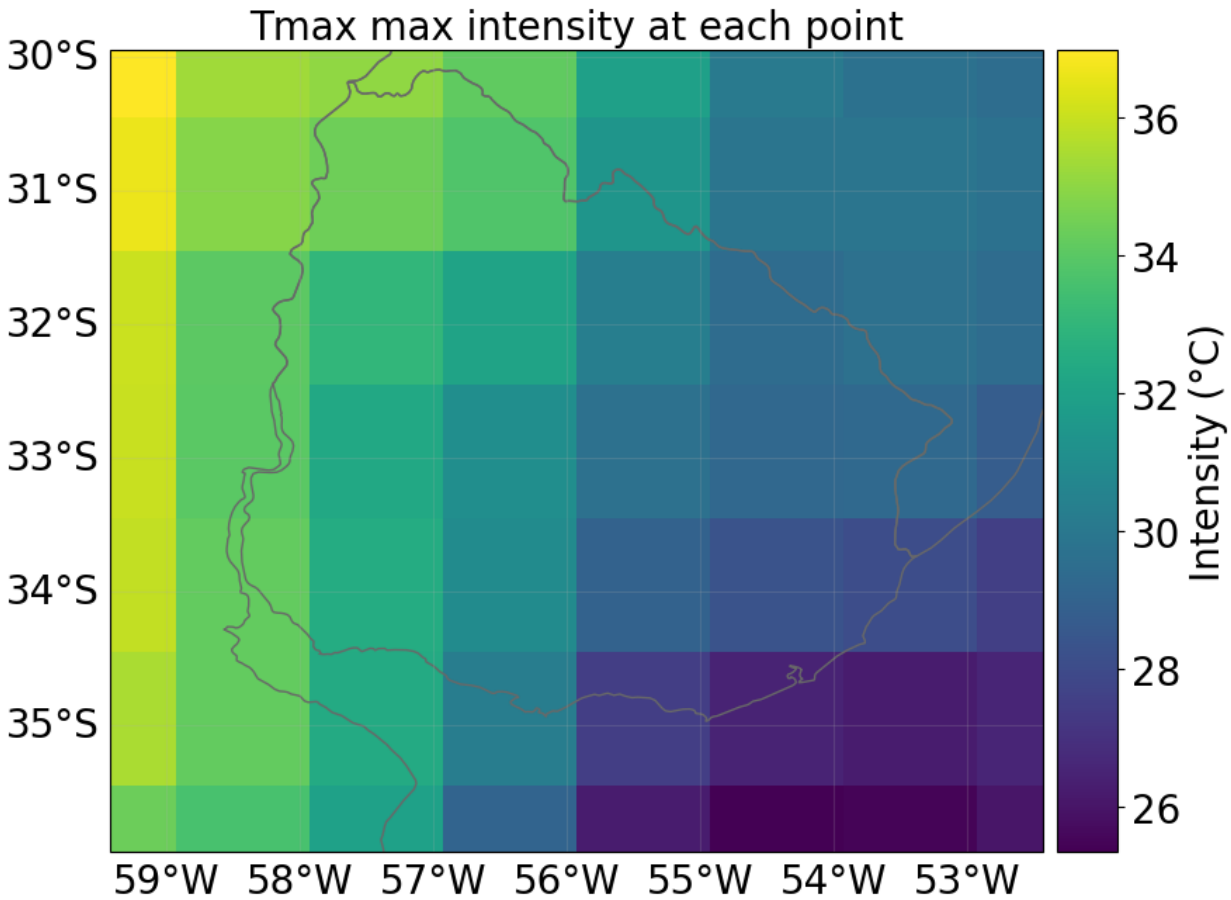
(continues on next page)

(continued from previous page)

```
  'member47',
  'member48',
  'member48',
  'member48',
  'member49',
  'member49',
  'member49'])
```

```
# load an example hazard
initiation_year = year_list[0]
initiation_month_str = f"{month_name_to_number(initiation_month[0]):02d}"

forecast_month = int(forecast.valid_period_str[-2:])
forecast_year = (
    initiation_year + 1
    if int(initiation_month_str) > forecast_month
    else initiation_year
)

path_to_hazard = forecast.get_pipeline_path(
    initiation_year, initiation_month_str, "hazard"
)
haz = Hazard.from_hdf5(path_to_hazard)

if haz:
    available_dates = sorted(set(haz.date))
    readable_dates = [
        dt.datetime.fromordinal(d).strftime("%Y-%m-%d") for d in available_dates
    ]
    print("Available Dates Across Members:", readable_dates)

    target_date = dt.datetime(
        forecast_year, forecast_month, 1
    ).toordinal()  # Look for the first day of the last forecast month
    closest_date = min(available_dates, key=lambda x: abs(x - target_date))
    closest_date_str = dt.datetime.fromordinal(closest_date).strftime("%Y-%m-%d")

    print(f"Selected Date for Plotting: {closest_date_str}")
    haz.select(date=[closest_date, closest_date]).plot_intensity(event=0,␣
↪smooth=False)

else:
    print("No hazard data found for the selected period.")
```

```
Available Dates Across Members: ['2022-12-01', '2023-01-01', '2023-02-01']
Selected Date for Plotting: 2023-02-01
```



Now you have a `Hazard` object that you can use in your specific impact assessment. In addition, you also have access to daily and monthly index estimates, along with ensemble statistics. Of course, the original hourly data of the climate variables related to your index of interest is also available, including their daily statistics.

If you would like to explore more advanced examples, please visit the package repository. There, you will find additional Python notebooks as well as links to plug-and-play Google Colab notebooks demonstrating the full capabilities of the package.

Resources

- Copernicus Seasonal Forecast Tools package
- Copernicus Seasonal Forecast Tools documentation
- Copernicus Seasonal Forecast Tools demo
- Copernicus Seasonal Forecast Tools extended demostration

Additional resources:

- U-CLIMADAPT Project
- Seasonal forecast daily and subdaily data on single levels
- Copernicus Climate Data Store
- CLIMADA Documentation

2.4 Exposures Tutorials

These guides present the *Exposures* class, as the main object to handle exposure data, as well as the *LitPop* subclass which allows to estimate exposure using nightlight intensity and population count data. We also show how to handle polygons or lines with CLIMADA.

2.4.1 Exposures class

What is an exposure?

Exposure describes the set of assets, people, livelihoods, infrastructures, etc. within an area of interest in terms of their geographic location, their value etc.; in brief - everything potentially exposed to hazards.

What options does CLIMADA offer for me to create an exposure?

CLIMADA has an `Exposures` class for this purpuse. An `Exposures` instance can be filled with your own data, or loaded from available default sources implemented through some Exposures-type classes from CLIMADA. If you have your own data, they can be provided in the formats of a `pandas.DataFrame`, a `geopandas.GeoDataFrame` or simply an `Excel` file. If you didn't collect your own data, exposures can be generated on the fly using CLIMADA's *LitPop*, BlackMarble or OpenStreetMap modules. See the respective tutorials to learn what exactly they contain and how to use them.

What does an exposure look like in CLIMADA?

An exposure is represented in the class `Exposures`, which contains a geopandas GeoDataFrame that is accessible through the `Exposures.data` attribute. A "geometry" column is initialized in the `GeoDataFrame` of the `Exposures` object, other columns are optional at first but some have to be present or make a difference when it comes to do calculations. Apart from these special columns the data frame may contain additional columns, they will simply be ignored in the context of CLIMADA.

The full list of meaningful columns is this:

Column	Data Type	Description	Meaningful in	Optional
geometry	Point	the geometry column of the `GeoDataFrame`, i.e., latitude (y) and longitude (x)	centroids assignment	-
value	float	a value for each exposure	impact calculation	✓*
impf_	int	impact functions ids for hazard types.important attribute, since it relates the exposures to the hazard by specifying the impf_act functions.Ideally it should be set to the specific hazard (e.g. `impf_TC`) so that different hazards can be setin the same Exposures (e.g. `impf_TC` and `impf_FL`).	impact calculation	✓*
centr	int	centroids index for hazard type.There might be different hazards defined: centr_TC, centr_FL, …Computed in method `assign_centroids()`	impact calculation	✓*
deduct	float	deductible value for each exposure. Used for insurance	impact calculation	✓
cover	float	cover value for each exposure. Used for insurance	impact calculation	✓
region_	int	region id (e.g. country ISO code) for each exposure	aggregation	✓
category_	int	category id (e.g. building code) for each exposure	aggregation	✓

*) an Exposures object is valid without such a column, but it's required for impact calculation

Apart from `data` the `Exposures` object has the following attributes and properties:

Attributes	Data Type	Description
`description`	str	describing origin and content of the exposures data
`ref_year`	int	reference year
`value_unit`	str	unit of the exposures' values

Properties	Data Type	Description
`geometry`	numpy.array[Point]	array of geometry values
`crs`	pyproj.CRS	coordinate reference system, see GeoDataFrame.crs
`latitude`	numpy.array[float]	array of latitude values
`longitude`	numpy.array[float]	array of longitude values
`region_id`	numpy.array[int]	array of regeion_id values
`category_id`	numpy.array[int]	array of category_id values
`cover`	numpy.array[float]	array of cover values
`deductible`	numpy.array[float]	array of cover values

Defining exposures from your own data

The essential structure of an exposure is similar, irrespective of the data type you choose to provide: As mentioned in the introduction, the key variables to be provided are `latitude`s, `longitude`s and `value`s of your exposed assets. While not mandatory, but very useful to provide for the impact calculation at later stages: the impact function id (see `impf_*` in the table above). The following examples will walk you through how to specifiy those four variables, and demonstrate the use of a few more optional parameters on the go.

Exposures from plain data

```
import numpy as np
from climada.entity import Exposures

latitude = [1, 2, 3] * 3
longitude = [4] * 3 + [5] * 3 + [6] * 3
exp_arr = Exposures(
    lat=latitude,  # list or array
    lon=longitude,  # instead of lat and lon one can provide an array of Points␣
↪through the geometry argument
    value=np.random.random_sample(len(latitude)),  # a list or an array of floats
    value_unit="CHF",
    crs="EPSG:7316",  # different formats are possible
    description="random values in a square",
    data={
        "region_id": 1,
        "impf_": range(len(latitude)),
    },  # data can also be an array or a data frame
)
print(exp_arr)
```

```
description: random values in a square
ref_year: 2018
value_unit: CHF
crs: EPSG:7316
data: (9 entries)
   region_id  impf_             geometry     value
0          1      0  POINT (4.000 1.000)  0.035321
1          1      1  POINT (4.000 2.000)  0.570256
2          1      2  POINT (4.000 3.000)  0.927632
3          1      3  POINT (5.000 1.000)  0.805402
4          1      4  POINT (5.000 2.000)  0.236179
```

(continues on next page)

(continued from previous page)

```
5          1      5  POINT (5.000 3.000)  0.848296
6          1      6  POINT (6.000 1.000)  0.520281
7          1      7  POINT (6.000 2.000)  0.036442
8          1      8  POINT (6.000 3.000)  0.780934
```

Exposures from a pandas DataFrame

In case you are unfamiliar with the data structure, check out the pandas DataFrame documentation.

```
import numpy as np
from pandas import DataFrame
from climada.entity import Exposures

# Fill a pandas DataFrame with the 3 mandatory variables (latitude, longitude, value)␣
↪for a number of assets (10'000).
# We will do this with random dummy data for purely illustrative reasons:
exp_df = DataFrame()
n_exp = 100 * 100
# provide value
exp_df["value"] = np.random.random_sample(n_exp)
# provide latitude and longitude
lat, lon = np.mgrid[
    15 : 35 : complex(0, np.sqrt(n_exp)), 20 : 40 : complex(0, np.sqrt(n_exp))
]
exp_df["latitude"] = lat.flatten()
exp_df["longitude"] = lon.flatten()
```

```
# For each exposure entry, specify which impact function should be taken for which␣
↪hazard type.
# In this case, we only specify the IDs for tropical cyclone (TC); here, each␣
↪exposure entry will be treated with
# the same impact function: the one that has ID '1':
# Of course, this will only be relevant at later steps during impact calculations.
exp_df["impf_TC"] = np.ones(n_exp, int)
exp_df
```

```
         value  latitude  longitude  impf_TC
0     0.533764      15.0  20.000000        1
1     0.995993      15.0  20.202020        1
2     0.603523      15.0  20.404040        1
3     0.754253      15.0  20.606061        1
4     0.305066      15.0  20.808081        1
...        ...       ...        ...      ...
9995  0.482416      35.0  39.191919        1
9996  0.069044      35.0  39.393939        1
9997  0.116560      35.0  39.595960        1
9998  0.239856      35.0  39.797980        1
9999  0.099568      35.0  40.000000        1

[10000 rows x 4 columns]
```

```
# Generate Exposures from the pandas DataFrame. This step converts the DataFrame into
# a CLIMADA Exposures instance!
exp = Exposures(exp_df)
print(f"exp has the type: {type(exp)}")
print(f"and contains a GeoDataFrame exp.gdf: {type(exp.gdf)}\n")
```

```
exp has the type: <class 'climada.entity.exposures.base.Exposures'>
and contains a GeoDataFrame exp.gdf: <class 'geopandas.geodataframe.GeoDataFrame'>
```

```
# let's have a look at the Exposures instance we created!
print(exp)
```

```
description: None
ref_year: 2018
value_unit: USD
crs: EPSG:4326
data: (10000 entries)
         value  impf_TC                   geometry
0     0.533764        1  POINT (20.00000 15.00000)
1     0.995993        1  POINT (20.20202 15.00000)
2     0.603523        1  POINT (20.40404 15.00000)
3     0.754253        1  POINT (20.60606 15.00000)
9996  0.069044        1  POINT (39.39394 35.00000)
9997  0.116560        1  POINT (39.59596 35.00000)
9998  0.239856        1  POINT (39.79798 35.00000)
9999  0.099568        1  POINT (40.00000 35.00000)
```

Exposures from a geopandas GeoDataFrame

In case you are unfamiliar with with data structure, check out the geopandas GeoDataFrame documentation. The main difference to the example above (pandas DataFrame) is that, while previously, we provided latitudes and longitudes which were then converted to a `geometry` GeoSeries using the `set_geometry_points` method, GeoDataFrames alread come with a defined `geometry` GeoSeries. In this case, we take the geometry info and use the `set_lat_lon` method to explicitly provide latitudes and longitudes. This example focuses on data with POINT geometry, but in principle, other geometry types (such as POLYGON and MULTIPOLYGON) would work as well.

```
import numpy as np
import geopandas as gpd
from climada.entity import Exposures

# Read spatial info from an external file into GeoDataFrame
world = gpd.read_file(gpd.datasets.get_path("naturalearth_cities"))
```

```
C:\Users\me\AppData\Local\Temp\ipykernel_31104\2272990317.py:6: FutureWarning: The␣
↪geopandas.dataset module is deprecated and will be removed in GeoPandas 1.0. You␣
↪can get the original 'naturalearth_cities' data from https://www.naturalearthdata.
↪com/downloads/110m-cultural-vectors/.
  world = gpd.read_file(gpd.datasets.get_path('naturalearth_cities'))
```

```
# Generate Exposures: value, latitude and longitude for each exposure entry.
world["value"] = np.arange(n_exp)
```

(continues on next page)

(continued from previous page)

```
# Convert GeoDataFrame into Exposure instance
exp_gpd = Exposures(world)
print(exp_gpd)
```

```
description: None
ref_year: 2018
value_unit: USD
crs: EPSG:4326
data: (243 entries)
             name     value                     geometry
0    Vatican City  0.876947    POINT (12.45339 41.90328)
1      San Marino  0.895454    POINT (12.44177 43.93610)
2           Vaduz  0.373366     POINT (9.51667 47.13372)
3         Lobamba  0.422729   POINT (31.20000 -26.46667)
239     São Paulo  0.913955  POINT (-46.62697 -23.55673)
240        Sydney  0.514479  POINT (151.21255 -33.87137)
241     Singapore  0.830635    POINT (103.85387 1.29498)
242     Hong Kong  0.764571   POINT (114.18306 22.30693)
```

```
# For each exposure entry, specify which impact function should be taken for which␣
↪hazard type.
# In this case, we only specify the IDs for tropical cyclone (TC); here, each␣
↪exposure entry will be treated with
# the same impact function: the one that has ID '1':
# Of course, this will only be relevant at later steps during impact calculations.
exp_gpd.data["impf_TC"] = np.ones(world.shape[0], int)
print(exp_gpd)
```

```
description: None
ref_year: 2018
value_unit: USD
crs: EPSG:4326
data: (243 entries)
             name     value                     geometry  impf_TC
0    Vatican City  0.876947    POINT (12.45339 41.90328)        1
1      San Marino  0.895454    POINT (12.44177 43.93610)        1
2           Vaduz  0.373366     POINT (9.51667 47.13372)        1
3         Lobamba  0.422729   POINT (31.20000 -26.46667)        1
239     São Paulo  0.913955  POINT (-46.62697 -23.55673)        1
240        Sydney  0.514479  POINT (151.21255 -33.87137)        1
241     Singapore  0.830635    POINT (103.85387 1.29498)        1
242     Hong Kong  0.764571   POINT (114.18306 22.30693)        1
```

The fact that `Exposures` is built around a `geopandas.GeoDataFrame` offers all the useful functionalities that come with the package. The following examples showcase only a few of those.

```
# Example 1: extract data in a region: latitudes between -5 and 5
sel_exp = exp_gpd.copy()  # to keep the original exp_gpd Exposures data
sel_exp.data = sel_exp.data.cx[:, -5:5]

print("\n" + "sel_exp contains a subset of the original data")
sel_exp.data
```

```
sel_exp contains a subset of the original data
```

```
                    name     value  impf_TC                     geometry
11                Tarawa  0.107688        1   POINT (173.01757 1.33819)
15                Kigali  0.218687        1   POINT (30.05859 -1.95164)
17                  Juba  0.763743        1    POINT (31.58003 4.82998)
31             Putrajaya  0.533607        1   POINT (101.69504 2.93252)
37             Bujumbura  0.127881        1   POINT (29.36001 -3.37609)
58               Kampala  0.079019        1    POINT (32.58138 0.31860)
75             Mogadishu  0.696766        1    POINT (45.36473 2.06863)
88                 Quito  0.212070        1  POINT (-78.50200 -0.21304)
93                Malabo  0.088459        1     POINT (8.78328 3.75002)
99            Libreville  0.929139        1     POINT (9.45796 0.38539)
108          Brazzaville  0.795766        1   POINT (15.28274 -4.25724)
113  Bandar Seri Begawan  0.655856        1   POINT (114.93328 4.88333)
116                Bangui  0.398002        1    POINT (18.55829 4.36664)
117               Yaoundé  0.240599        1    POINT (11.51470 3.86865)
134              Victoria  0.956208        1   POINT (55.44999 -4.61663)
135              São Tomé  0.726704        1     POINT (6.72965 0.33747)
138                  Malé  0.996017        1    POINT (73.50890 4.17204)
158         Kuala Lumpur  0.880473        1   POINT (101.68870 3.13980)
201             Kinshasa  0.074387        1   POINT (15.31303 -4.32778)
228              Nairobi  0.297170        1   POINT (36.81471 -1.28140)
230               Bogota  0.420891        1   POINT (-74.08529 4.59837)
241            Singapore  0.830635        1   POINT (103.85387 1.29498)
```

```
# Example 2: extract data in a polygon
from shapely.geometry import Polygon

sel_polygon = exp_gpd.copy()  # to keep the original exp_gpd Exposures data

poly = Polygon([(0, -10), (0, 10), (10, 5)])
sel_polygon.data = sel_polygon.gdf[sel_polygon.gdf.intersects(poly)]

# Let's have a look. Again, the sub-selection is a GeoDataFrame!
print("\n" + "sel_exp contains a subset of the original data")
sel_polygon.data
```

```
sel_exp contains a subset of the original data
```

```
           name     value  impf_TC                 geometry
36   Porto-Novo  0.573619        1  POINT (2.61663 6.48331)
46         Lomé  0.176892        1  POINT (1.22081 6.13388)
93       Malabo  0.088459        1  POINT (8.78328 3.75002)
123     Cotonou  0.441703        1  POINT (2.40435 6.36298)
135    São Tomé  0.726704        1  POINT (6.72965 0.33747)
225       Lagos  0.990135        1  POINT (3.38959 6.44521)
```

```
# Example 3: change coordinate reference system
# use help to see more options: help(sel_exp.to_crs)
sel_polygon.to_crs(epsg=3395, inplace=True)
```

(continues on next page)

(continued from previous page)

```
print("\n" + "the crs has changed to " + str(sel_polygon.crs))
print(
    "the values for latitude and longitude are now according to the new coordinate␣
↪system: "
)
sel_polygon.data
```

```
the crs has changed to EPSG:3395
the values for latitude and longitude are now according to the new coordinate system:
```

```
           name     value  impf_TC                       geometry
36   Porto-Novo  0.573619        1  POINT (291281.418 718442.692)
46         Lomé  0.176892        1  POINT (135900.092 679566.331)
93       Malabo  0.088459        1  POINT (977749.979 414955.553)
123     Cotonou  0.441703        1  POINT (267651.551 705052.049)
135    São Tomé  0.726704        1   POINT (749141.190 37315.322)
225       Lagos  0.990135        1  POINT (377326.898 714202.107)
```

```
# Example 4: concatenate exposures
exp_all = Exposures.concat([sel_polygon, sel_exp.to_crs(epsg=3395)])

# the output is of type Exposures
print("exp_all type and number of rows:", type(exp_all), exp_all.gdf.shape[0])
print("number of unique rows:", exp_all.gdf.drop_duplicates().shape[0])

# NaNs will appear in the missing values
exp_all.data.tail()
```

```
exp_all type and number of rows: <class 'climada.entity.exposures.base.Exposures'> 28
number of unique rows: 26
```

```
            name     value  impf_TC                        geometry
23  Kuala Lumpur  0.880473        1  POINT (11319934.225 347356.996)
24      Kinshasa  0.074387        1  POINT (1704638.257 -479002.730)
25       Nairobi  0.297170        1  POINT (4098194.882 -141701.948)
26        Bogota  0.420891        1  POINT (-8247136.736 509015.405)
27     Singapore  0.830635        1  POINT (11560960.460 143203.754)
```

Exposures of any file type supported by Geopandas and Pandas

Geopandas can read almost any vector-based spatial data format including ESRI shapefile, GeoJSON files and more, see readers geopandas. Pandas supports formats such as csv, html or sql; see readers pandas. Using the corresponding readers, `DataFrame` and `GeoDataFrame` can be filled and provided to `Exposures` following the previous examples.

Exposures from an excel file

If you manually collect exposure data, Excel may be your preferred option. In this case, it is easiest if you format your data according to the structure provided in the template `climada_python/climada/data/system/entity_template.xlsx`, in the sheet `assets`.

```
import pandas as pd
from climada.util.constants import ENT_TEMPLATE_XLS
from climada.entity import Exposures

# Read your Excel file into a pandas DataFrame (we will use the template example for␣
↪this demonstration):
file_name = ENT_TEMPLATE_XLS
exp_templ = pd.read_excel(file_name)

# Let's have a look at the data:
print("exp_templ is a DataFrame:", str(type(exp_templ)))
print("exp_templ looks like:")
exp_templ.head()
```

```
exp_templ is a DataFrame: <class 'pandas.core.frame.DataFrame'>
exp_templ looks like:
```

```
    latitude  longitude         value  deductible         cover  region_id  \
0  26.933899 -80.128799  1.392750e+10           0  1.392750e+10          1
1  26.957203 -80.098284  1.259606e+10           0  1.259606e+10          1
2  26.783846 -80.748947  1.259606e+10           0  1.259606e+10          1
3  26.645524 -80.550704  1.259606e+10           0  1.259606e+10          1
4  26.897796 -80.596929  1.259606e+10           0  1.259606e+10          1

   category_id  impf_TC  centr_TC  impf_FL  centr_FL
0            1        1         1        1         1
1            1        1         2        1         2
2            1        1         3        1         3
3            1        1         4        1         4
4            1        1         5        1         5
```

As we can see, the general structure is the same as always: the exposure has `latitude`, `longitude` and `value` columns. Further, this example specified several impact function ids: some for Tropical Cyclones (`impf_TC`), and some for Floods (`impf_FL`). It also provides some meta-info (`region_id`, `category_id`) and insurance info relevant to the impact calculation in later steps (`cover`, `deductible`).

```
# Generate an Exposures instance from the dataframe.
exp_templ = Exposures(exp_templ)
print("\n" + "exp_templ is now an Exposures:", exp_templ)
```

```
exp_templ is now an Exposures: description: None
ref_year: 2018
value_unit: USD
crs: EPSG:4326
data: (24 entries)
           value  deductible         cover  region_id  category_id  impf_TC  \
0   1.392750e+10           0  1.392750e+10          1            1        1
1   1.259606e+10           0  1.259606e+10          1            1        1
2   1.259606e+10           0  1.259606e+10          1            1        1
3   1.259606e+10           0  1.259606e+10          1            1        1
20  1.259760e+10           0  1.259760e+10          1            1        1
21  1.281454e+10           0  1.281454e+10          1            1        1
```

(continues on next page)

(continued from previous page)

```
22  1.262176e+10           0  1.262176e+10          1            1        1
23  1.259754e+10           0  1.259754e+10          1            1        1

    centr_TC  impf_FL  centr_FL                    geometry
0          1        1         1  POINT (-80.12880 26.93390)
1          2        1         2  POINT (-80.09828 26.95720)
2          3        1         3  POINT (-80.74895 26.78385)
3          4        1         4  POINT (-80.55070 26.64552)
20        21        1        21  POINT (-80.06858 26.71255)
21        22        1        22  POINT (-80.09070 26.66490)
22        23        1        23  POINT (-80.12540 26.66470)
23        24        1        24  POINT (-80.15140 26.66315)
```

Exposures from a raster file

Last but not least, you may have your exposure data stored in a raster file. Raster data may be read in from any file-type supported by rasterio.

```
from rasterio.windows import Window
from climada.util.constants import HAZ_DEMO_FL
from climada.entity import Exposures

# We take an example with a dummy raster file (HAZ_DEMO_FL), running the method set_
↪from_raster directly loads the
# necessary info from the file into an Exposures instance.
exp_raster = Exposures.from_raster(HAZ_DEMO_FL, window=Window(10, 20, 50, 60))
# There are several keyword argument options that come with the set_from_raster␣
↪method (such as
# specifying a window, if not the entire file should be read, or a bounding box.␣
↪Check them out.
```

```
2024-10-04 17:19:03,632 - climada.util.coordinates - INFO - Reading C:\Users\me\
↪climada\demo\data\SC22000_VE__M1.grd.gz
```

```
exp_raster.derive_raster()
```

```
2024-10-04 17:19:03,725 - climada.util.coordinates - INFO - Raster from resolution 0.
↪009000000000000341 to 0.009000000000000341.
```

```
{'crs': <Geographic 2D CRS: EPSG:4326>
 Name: WGS 84
 Axis Info [ellipsoidal]:
 - Lat[north]: Geodetic latitude (degree)
 - Lon[east]: Geodetic longitude (degree)
 Area of Use:
 - name: World.
 - bounds: (-180.0, -90.0, 180.0, 90.0)
 Datum: World Geodetic System 1984 ensemble
 - Ellipsoid: WGS 84
 - Prime Meridian: Greenwich,
 'height': 60,
```

(continues on next page)

(continued from previous page)

```
 'width': 50,
 'transform': Affine(0.009000000000000341, 0.0, -69.2471495969998,
        0.0, -0.009000000000000341, 10.248220966978932)}
```

Loading CLIMADA-generated exposure files or generating new ones

In case you already have a CLIMADA-generated file containing `Exposures` info, you can of course load it back into memory. Most likely, the data format will either be of `.hdf5` or of `.mat`.

In case you neither have your own data, nor a CLIMADA-generated file, you can also create an exposure on the fly using one of the three CLIMADA-internal exposure generators: *LitPop*, BlackMarble or OpenStreetMap modules. The latter three are extensively described in their own, linked, tutorials.

```
# read  generated with the Python version with from_hdf5()
# note: for .mat data, use the method from_mat() analogously.
from climada.util.constants import EXP_DEMO_H5

exp_hdf5 = Exposures.from_hdf5(EXP_DEMO_H5)
print(exp_hdf5)
```

```
2024-10-04 17:19:04,888 - climada.entity.exposures.base - INFO - Reading C:\Users\me\
↪climada\demo\data\exp_demo_today.h5
description: None
ref_year: 2016
value_unit: USD
crs: EPSG:4326
data: (50 entries)
           value  impf_TC  deductible         cover  category_id  region_id  \
0   1.392750e+10        1         0.0  1.392750e+10            1        1.0
1   1.259606e+10        1         0.0  1.259606e+10            1        1.0
2   1.259606e+10        1         0.0  1.259606e+10            1        1.0
3   1.259606e+10        1         0.0  1.259606e+10            1        1.0
46  1.264524e+10        1         0.0  1.264524e+10            1        1.0
47  1.281438e+10        1         0.0  1.281438e+10            1        1.0
48  1.260291e+10        1         0.0  1.260291e+10            1        1.0
49  1.262482e+10        1         0.0  1.262482e+10            1        1.0

                      geometry
0   POINT (-80.12880 26.93390)
1   POINT (-80.09828 26.95720)
2   POINT (-80.74895 26.78385)
3   POINT (-80.55070 26.64552)
46  POINT (-80.11640 26.34907)
47  POINT (-80.08385 26.34635)
48  POINT (-80.24130 26.34802)
49  POINT (-80.15886 26.34796)
```

```
c:\Users\me\miniconda3\envs\climada_env\Lib\pickle.py:1718: UserWarning: Unpickling a␣
↪shapely <2.0 geometry object. Please save the pickle again; shapely 2.1 will not␣
↪have this compatibility.
  setstate(state)
```

Visualize Exposures

The method `plot_hexbin()` uses cartopy and matplotlib's hexbin function to represent the exposures values as 2d bins over a map. Configure your plot by fixing the different inputs of the method or by modifying the returned `matplotlib` figure and axes.

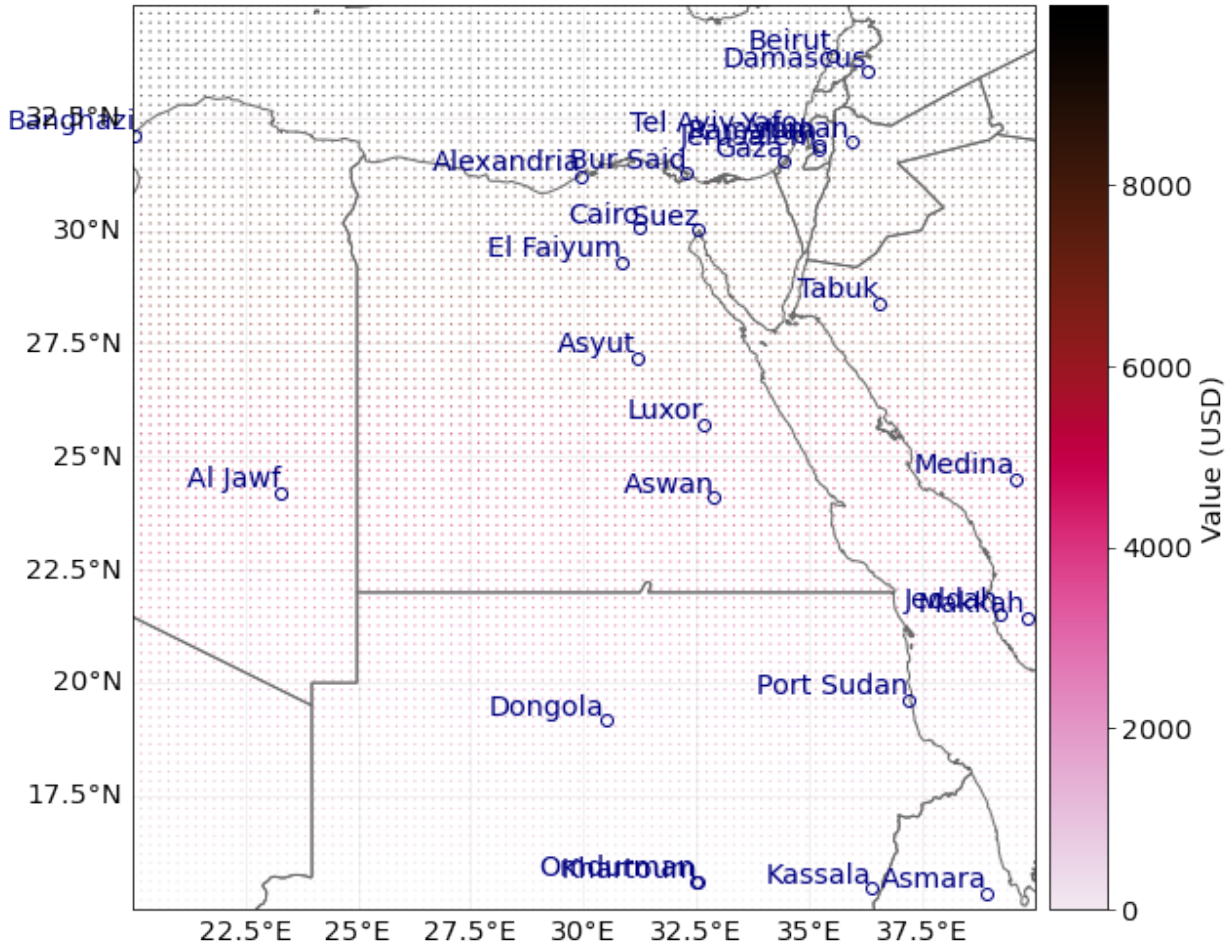
The method `plot_scatter()` uses cartopy and matplotlib's scatter function to represent the points values over a 2d map. As usal, it returns the figure and axes, which can be modify aftwerwards.

The method `plot_raster()` rasterizes the points into the given resolution. Use the `save_tiff` option to save the resulting `tiff` file and the `res_raster`option to re-set the raster's resolution.

Finally, the method `plot_basemap()` plots the scatter points over a satellite image using contextily library.

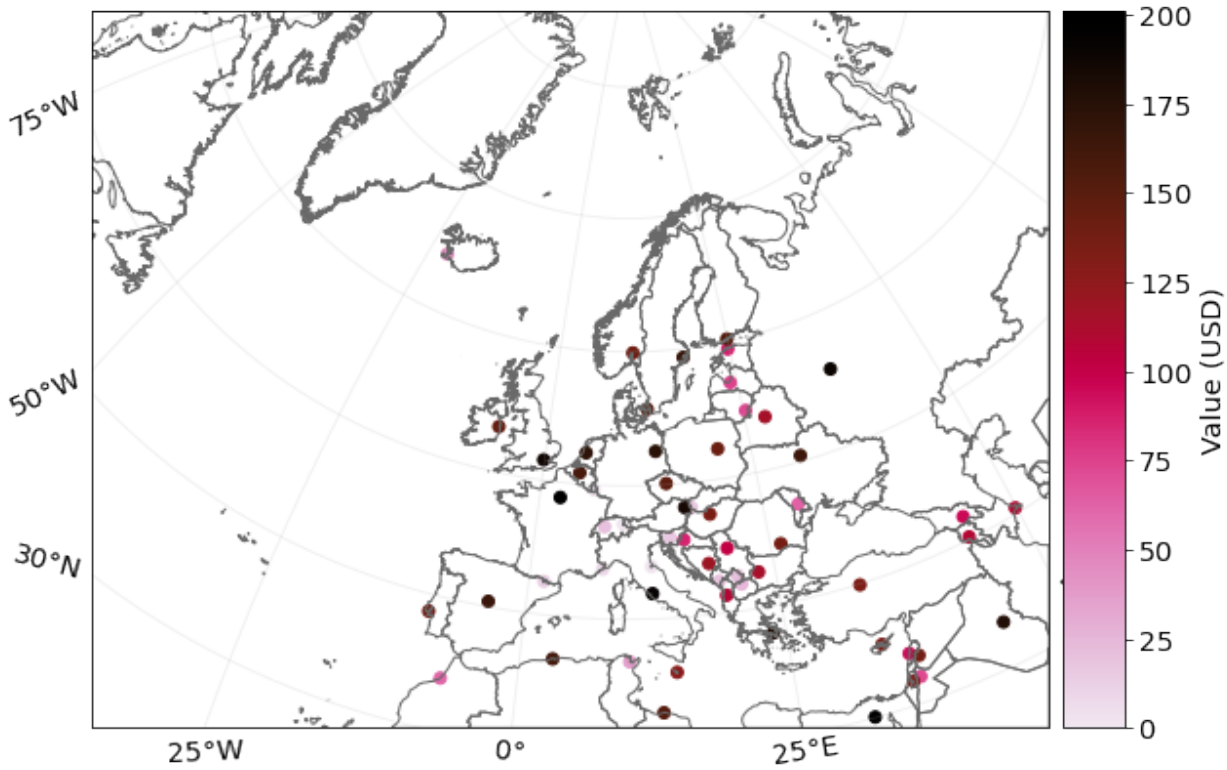
```
# Example 1: plot_hexbin method
print("Plotting exp_df.")
axs = exp.plot_hexbin();

# further methods to check out:
# axs.set_xlim(15, 45) to modify x-axis borders, axs.set_ylim(10, 40) to modify y-
↪axis borders
# further keyword arguments to play around with: pop_name, buffer, gridsize, ...
```

```
Plotting exp_df.
```



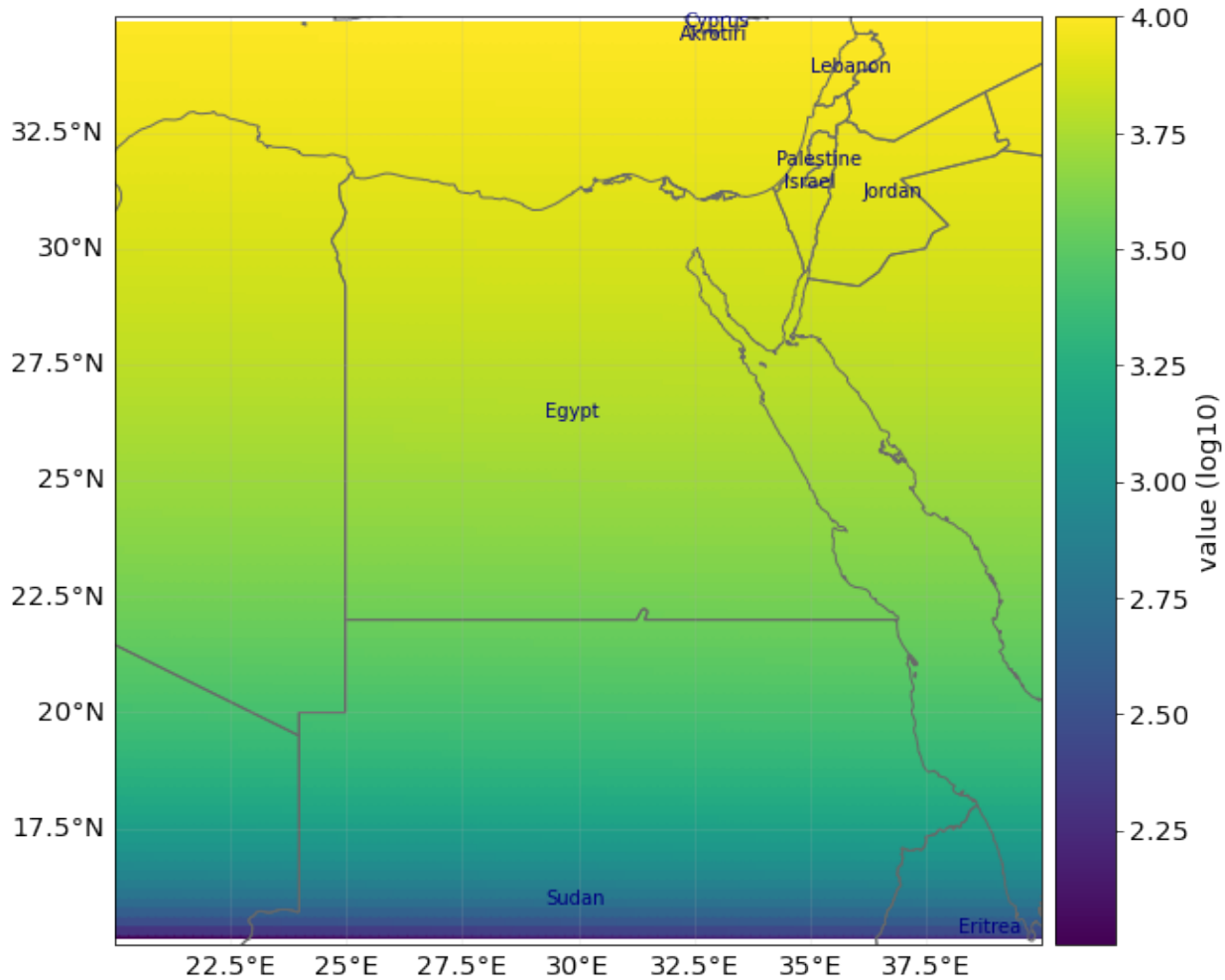
```
# Example 2: plot_scatter method

exp_gpd.to_crs("epsg:3035", inplace=True)
exp_gpd.plot_scatter(pop_name=False);
```

```
<GeoAxesSubplot:>
```

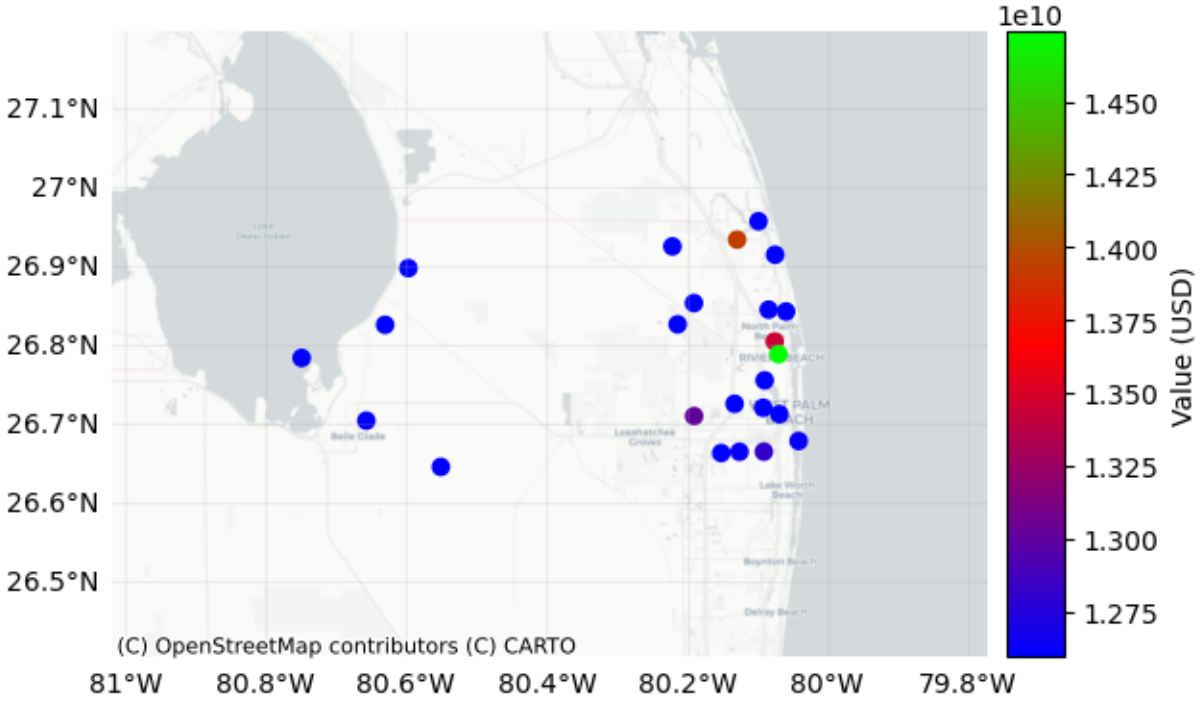


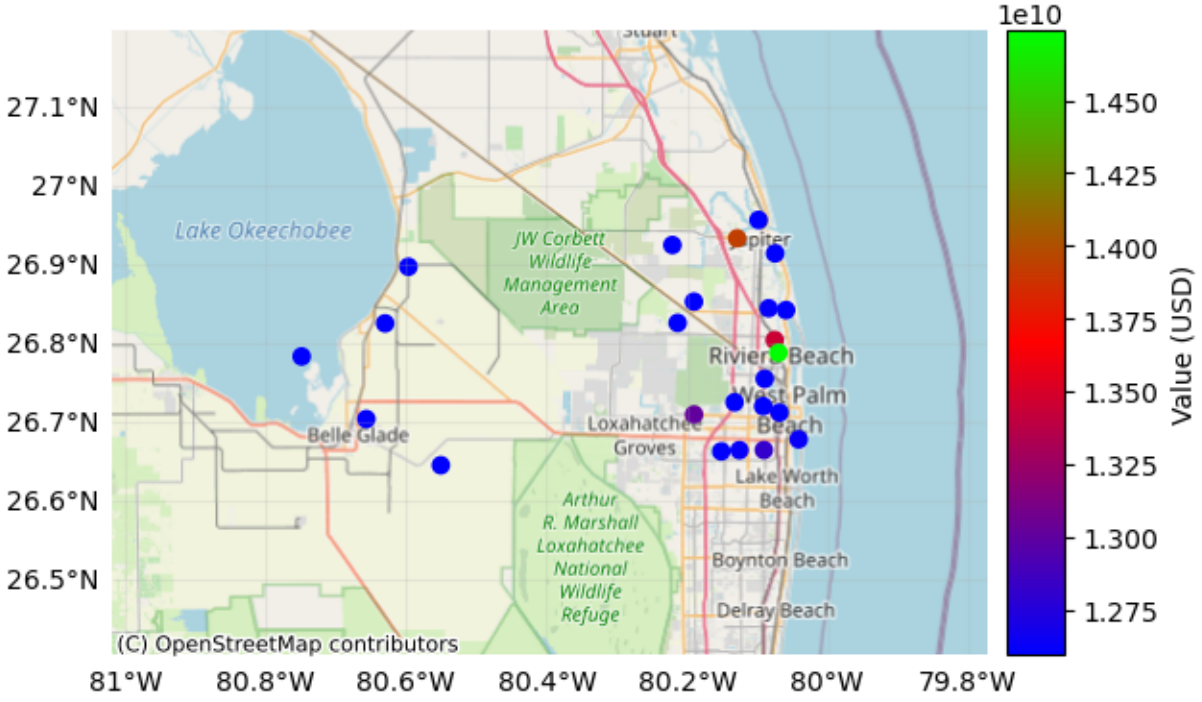

```
# Example 3: plot_raster method
from climada.util.plot import add_cntry_names  # use climada's plotting utilities

ax = exp.plot_raster()
# plot with same resolution as data
add_cntry_names(
    ax,
    [
        exp.gdf["longitude"].min(),
        exp.gdf["longitude"].max(),
        exp.gdf["latitude"].min(),
        exp.gdf["latitude"].max(),
    ],
)

# use keyword argument save_tiff='filepath.tiff' to save the corresponding raster in␣
↪tiff format
# use keyword argument raster_res='desired number' to change resolution of the raster.
```

```
2021-06-04 17:07:42,654 - climada.util.coordinates - INFO - Raster from resolution 0.
↪20202020202019355 to 0.20202020202019355.
```




```
# Example 4: plot_basemap method
import contextily as ctx

# select the background image from the available ctx.providers
ax = exp_templ.plot_basemap(buffer=30000, cmap="brg")
# using Positron from CartoDB
ax = exp_templ.plot_basemap(
    buffer=30000,
    cmap="brg",
    url=ctx.providers.OpenStreetMap.Mapnik,  # Using OpenStreetmap,
    zoom=9,
);  # select the zoom level of the map, affects the font size of labelled objects
```

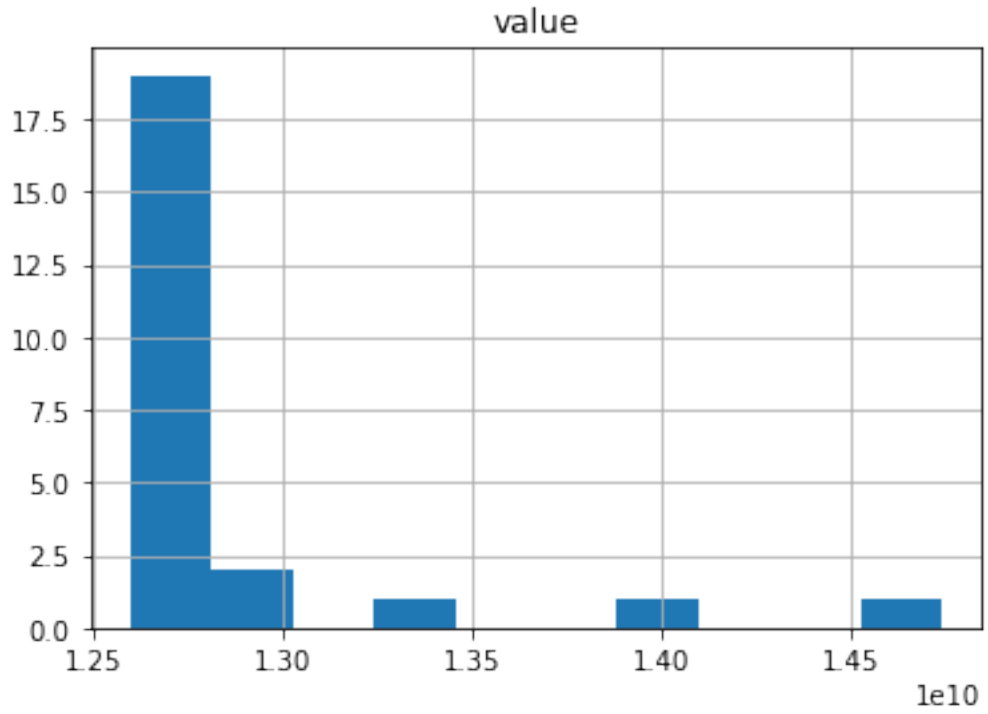




Since `Exposures` is a `GeoDataFrame`, any function for visualization from geopandas can be used. Check making maps and examples gallery.

```
# other visualization types
exp_templ.gdf.hist(column="value");
```

```
array([[<AxesSubplot:title={'center':'value'}>]], dtype=object)
```



Write (Save) Exposures

Exposures can be saved in any format available for `GeoDataFrame` (see fiona.supported_drivers) and `DataFrame` (pandas IO tools). Take into account that in many of these formats the metadata (e.g. variables `ref_year` and `value_unit`) will not be saved. Use instead the format hdf5 provided by `Exposures` methods `write_hdf5()` and `from_hdf5()` to handle all the data.

```
import fiona

fiona.supported_drivers
from climada import CONFIG

results = CONFIG.local_data.save_dir.dir()

# DataFrame save to csv format. geometry writen as string, metadata not saved!
exp_templ.gdf.to_csv(results.joinpath("exp_templ.csv"), sep="\t")
```

```
# write as hdf5 file
exp_templ.write_hdf5(results.joinpath("exp_temp.h5"))
```

Optionally use climada's save option to save it in pickle format. This allows fast to quickly restore the object in its current state and take up your work right were you left it the next time. Note however, that pickle has a transient format and is not suitable for storing data persistently.

```
# save in pickle format
from climada.util.save import save
```

(continues on next page)

(continued from previous page)

```
# this generates a results folder in the current path and stores the output there
save("exp_templ.pkl.p", exp_templ)  # creates results folder and stores there
```

2.4.2 LitPop class

Introduction

`LitPop` is an *Exposures*-type class. It is used to initiate gridded exposure data with estimates of either asset value, economic activity or population based on nightlight intensity and population count data.

Background

The modeling of economic disaster risk on a global scale requires high-resolution maps of exposed asset values. We have developed a generic and scalable method to downscale national asset value estimates proportional to a combination of nightlight intensity ("Lit") and population data ("Pop").

Asset exposure value is disaggregated to the grid points proportionally to $Lit^m Pop^n$, computed at each grid cell:

$Lit^m Pop^n = Lit^m * Pop^n$, with $exponents = [m, n] \in \mathbb{R}^+$ (Default values are $m = n = 1$).

For more information please refer to the related publication (https://doi.org/10.5194/essd-12-817-2020) and data archive (https://doi.org/10.3929/ethz-b-000331316).

How to cite: Eberenz, S., Stocker, D., Röösli, T., and Bresch, D. N.: *Asset exposure data for global physical risk assessment*, Earth Syst. Sci. Data, 12, 817–833, https://doi.org/10.5194/essd-12-817-2020, 2020.

Input data

Note: All required data except for the population data from Gridded Population of the World (GPW) is downloaded automatically when an `LitPop.set_*` method is called.

Warning: Processing the data for the first time can take up huge amounts of RAM (>10 GB), depending on country or region size. Consider using the *wrapper function* of the *data API* to download readily computed LitPop exposure data for default values ($n = m = 1$) on demand.

Nightlight intensity

Black Marble annual composite of the VIIRS day-night band (Grayscale) at 15 arcsec resolution is downloaded from the NASA Earth Observatory: https://earthobservatory.nasa.gov/Features/NightLights (available for 2012 and 2016 at 15 arcsec resolution (~500m)). The first time a nightlight image is used, it is downloaded and stored locally. This might take some time.

Population count

Gridded Population of the World (GPW), v4: Population Count, v4.10, v4.11 or later versions (2000, 2005, 2010, 2015, 2020), available from http://sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse.

The GPW file of the year closest to the requested year (`reference_year`) is required. To download GPW data a (free) login for the NASA SEDAC website is required.

Direct download links are avilable, also for older versions, i.e.:

- v4.11: http://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/gpw-v4-population-count-rev11/gpw-v4-population-count-rev11_2015_30_sec_tif.zip
- v4.10: http://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/gpw-v4-population-count-rev10/gpw-v4-population-count-rev10_2015_30_sec_tif.zip,

- Overview over all versions of GPW v.4: https://beta.sedac.ciesin.columbia.edu/data/collection/gpw-v4/sets/browse

The population data from GWP needs to be downloaded manually as TIFF from this site and placed in the `SYSTEM_DIR` folder of your climada installation.

Downloading existing LitPop asset exposure data

The easiest way to download existing data is using the *wrapper function* of the *data API*.

Readily computed LitPop asset exposure data based on Lit^1Pop^1 for 224 countries, distributing produced capital / non-financial wealth of 2014 at a resolution of 30 arcsec can also be downloaded from the ETH Research Repository: https://doi.org/10.3929/ethz-b-000331316. The dataset contains gridded data for more than 200 countries as CSV files.

Attributes

The `LitPop` class inherits from *`Exposures`*. It adds the following attributes:

```
    exponents : Defining powers (m, n) with which nightlights and population go into␣
↪Lit**m * Pop**n.
    fin_mode : Socio-economic indicator to be used as total asset value for␣
↪disaggregation.
    gpw_version : Version number of GPW population data, e.g. 11 for v4.11
```

fin_mode

The choice of `fin_mode` is crucial. Implemented choices are:

- `'pc'`: produced capital (Source: World Bank), incl. manufactured or built assets such as machinery, equipment, and physical structures. The pc-data is stored in the subfolder *data/system/Wealth-Accounts_CSV/*. Source: https://datacatalog.worldbank.org/dataset/wealth-accounting
- `'pop'`: population count (source: GPW, same as gridded population)
- `'gdp'`: gross-domestic product (Source: World Bank)
- `'income_group'`: gdp multiplied by country's income group+1
- `'nfw'`: non-financial household wealth (Source: Credit Suisse)
- `'tw'`: total household wealth (Source: Credit Suisse)
- `'norm'`: normalized, total value of country or region is 1.
- `'none'`: None – LitPop per pixel is returned unchanged

Regarding the GDP (nominal GDP at current USD) and income group values, they are obtained from the World Bank using the pandas-datareader API. If a value is missing, the value of the closest year is considered. When no values are provided from the World Bank, we use the Natural Earth repository values.

Key Methods

- `from_countries`: set exposure for one or more countries, see section `from_countries` below.
- `from_nightlight_intensity`: wrapper around `from_countries` and `from_shape` to load nightlight data to exposure.
- `from_population`: wrapper around `from_countries` and `from_shape_population` to load pure population data to exposure. This can be used to initiate a population exposure set.
- `from_shape_and_countries`: given a shape and a list of countries, exposure is initiated for the countries and then cropped to the shape. See section *Set custom shapes* below.

- `from_shape`: given any shape or geometry and an estimate of total values, exposure is initiated for the shape directly. See section *Set custom shapes* below.

```
# Import class LitPop:
from climada.entity import LitPop
```

from_countries

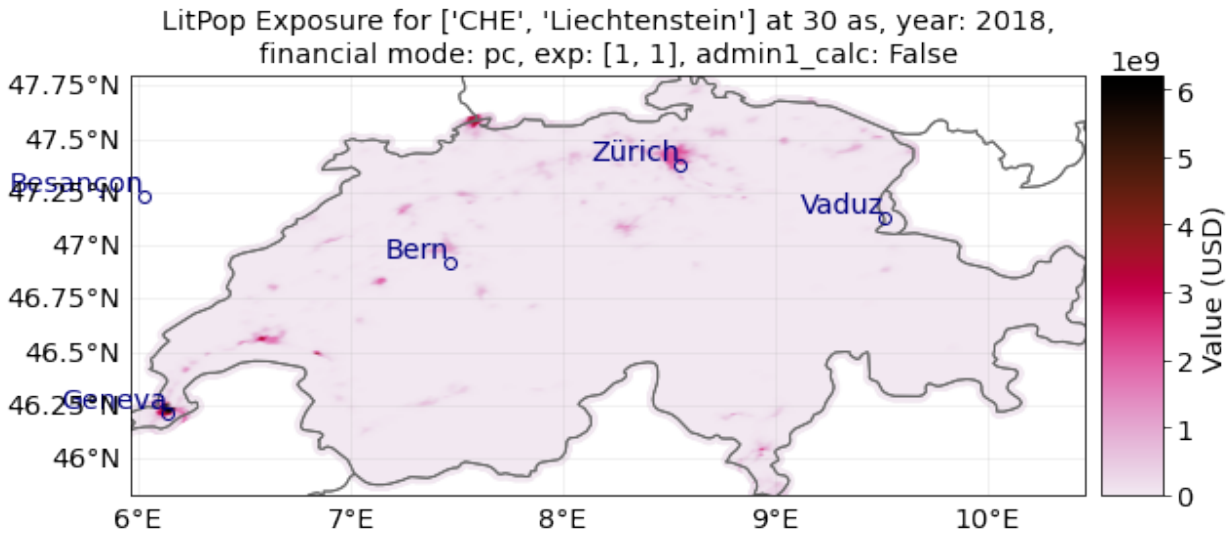
In the following, we will create exposure data sets and plots for a variety of countries, comparing different settings.

Default Settings

Per default, the exposure entity was initiated using the default parameters, i.e. a resolution of 30 arcsec, produced capital 'pc' as total asset value and using the exponents $(1, 1)$.

```
# Initiate a default LitPop exposure entity for Switzerland and Liechtenstein (ISO3-
↪Codes 'CHE' and 'LIE'):
try:
    exp = LitPop.from_countries(
        ["CHE", "Liechtenstein"]
    )  # you can provide either single countries or a list of countries
except FileExistsError as err:
    print(
        "Reason for error: The GPW population data has not been downloaded, c.f.␣
↪section 'Input data' above."
    )
    raise err
exp.plot_scatter()

# Note that `exp.gdf['region_id']` is a number identifying each country:
print("\n Region IDs (`region_id`) in this exposure:")
print(exp.gdf["region_id"].unique())
```

```
2021-10-19 17:03:20,108 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2021-10-19 17:03:23,876 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2021-10-19 17:03:23,953 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2021-10-19 17:03:24,476 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.

 Region IDs (`region_id`) in this exposure:
[756 438]
```



fin_mode, resolution and exponents

Instead on produced capital, we can also downscale other available macroeconomic indicators as estimates of asset value. The indicator can be set via the parameter `fin_mode`, either to 'pc', 'pop', 'gdp', 'income_group', 'nfw', 'tw', 'norm', or 'none'. See descriptions of each alternative above in the introduction.

We can also change the resolution via `res_arcsec` and the `exponents`.

The default resolution is 30 arcsec ≈ 1 km. A resolution of 3600 arcsec = 1 degree corresponds to roughly 110 km close to the equator.

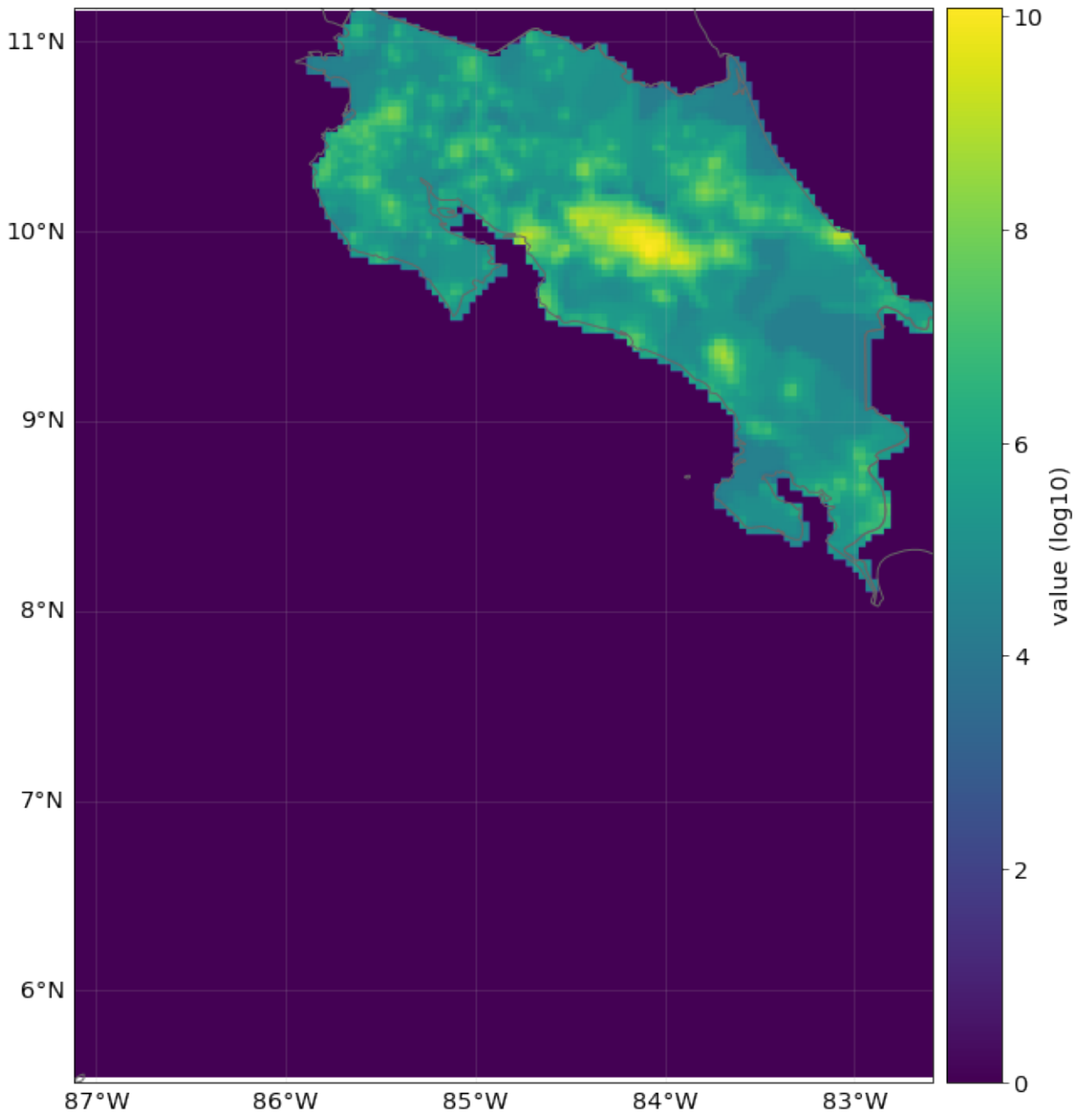
from_population

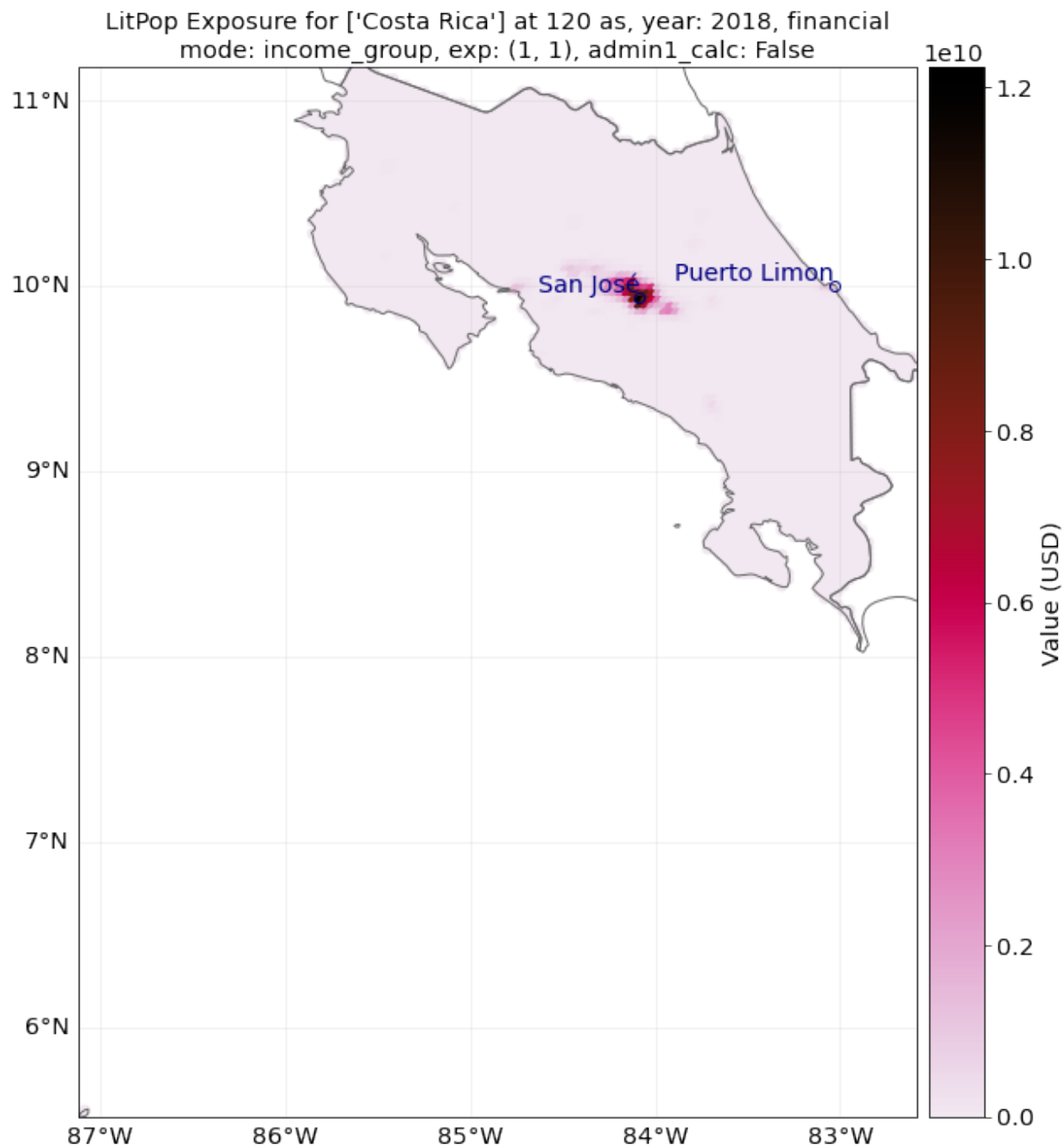
Let's initiate an exposure instance with the financial mode "income_group" and at a resolution of 120 arcsec (roughly 4 km).

```
# Initiate a LitPop exposure entity for Costa Rica with varied resolution, fin_mode,␣
↪and exponents:
exp = LitPop.from_countries(
    "Costa Rica", fin_mode="income_group", res_arcsec=120, exponents=(1, 1)
)  # change the parameters and see what happens...
# exp = LitPop.from_countries('Costa Rica', fin_mode='gdp', res_arcsec=90,␣
↪exponents=(3,0)) # example of variation
exp.plot_raster()
# note the log scale of the colorbar
exp.plot_scatter();
```

```
2021-10-19 17:03:26,671 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

```
<GeoAxesSubplot:title={'center':"LitPop Exposure for ['Costa Rica'] at 120 as, year:␣
↪2018, financial\nmode: income_group, exp: (1, 1), admin1_calc: False"}>
```


11°N
10°N
9°N
8°N
7°N
6°N
87°W
86°W
85°W
84°W
83°W
10
8
6
4
2
0
value (log10)



Reference year

Additionally, we can change the year our exposure is supposed to represent. For this, nightlight and population data are used that are closest to the requested years. Macroeconomic indicators like produced capital are interpolated from available data or scaled proportional to GDP.

Let's load a population exposure map for Switzerland in 2000 and 2021 with a resolution of 300 arcsec:

```
# You may want to check if you have downloaded dataset Gridded Population of the␣
↪World (GPW), v4: Population Count, v4.11
# (2000 and 2020) first
```

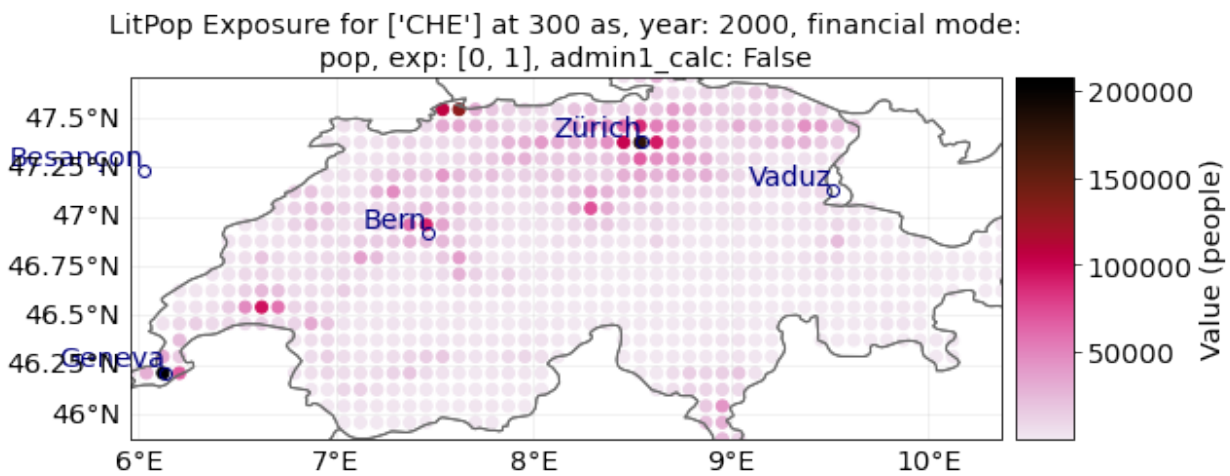
(continues on next page)

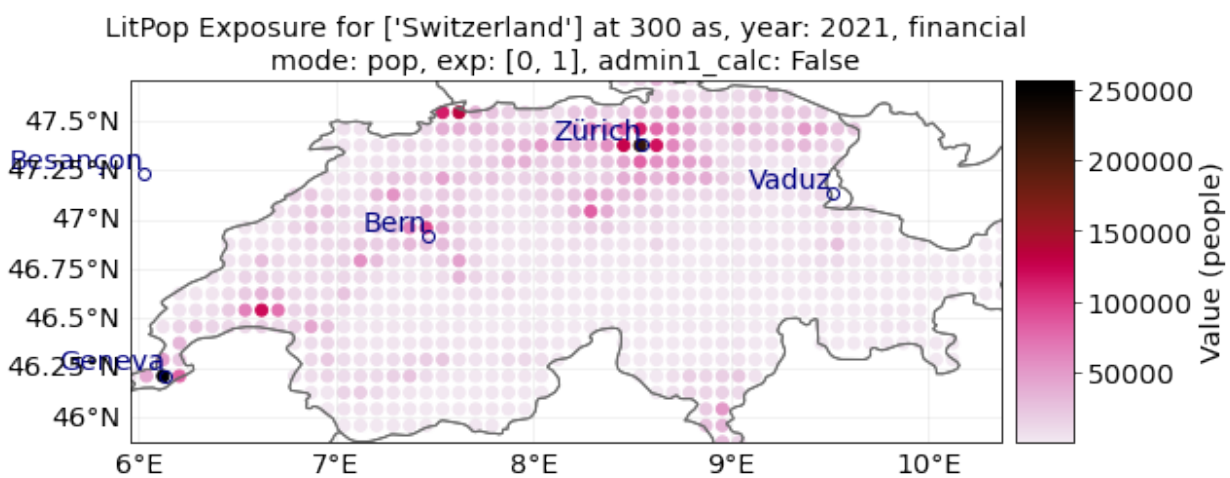
(continued from previous page)

```
pop_2000 = LitPop.from_countries(
    "CHE", fin_mode="pop", res_arcsec=300, exponents=(0, 1), reference_year=2000
)
# Alternatively, we ca use `from_population`:
pop_2021 = LitPop.from_population(
    countries="Switzerland", res_arcsec=300, reference_year=2021
)
# Since no population data for 2021 is available, the closest data point, 2020, is␣
↪used (see LOGGER.warning)
pop_2000.plot_scatter()
pop_2021.plot_scatter()
"""Note the difference in total values on the color bar."""
```

```
2021-10-19 17:03:31,884 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2021. Using nearest available year for GPW data: 2020
```

```
'Note the difference in total values on the color bar.'
```





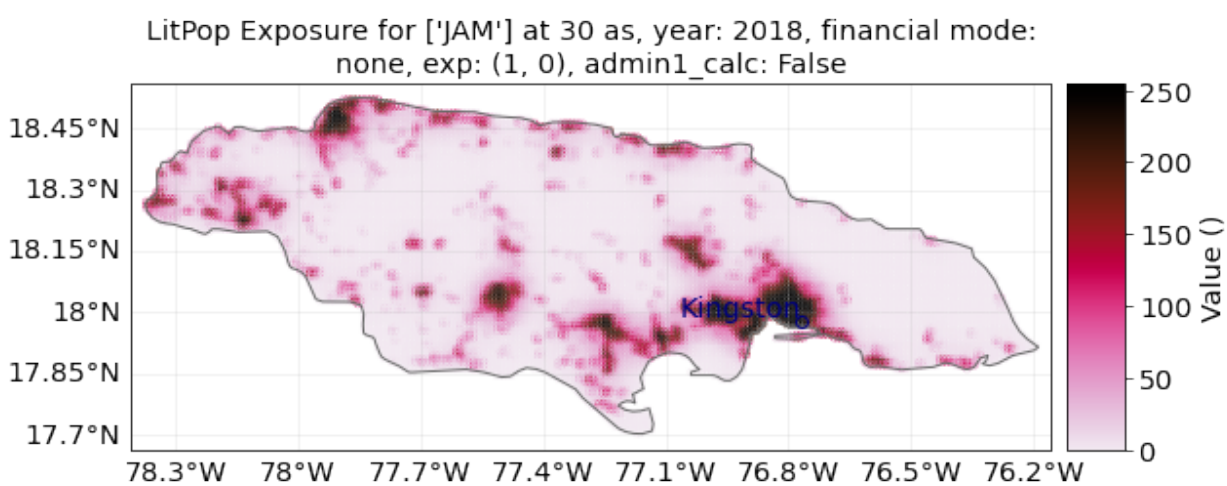
`from_nightlight_intensity` and `from_population`

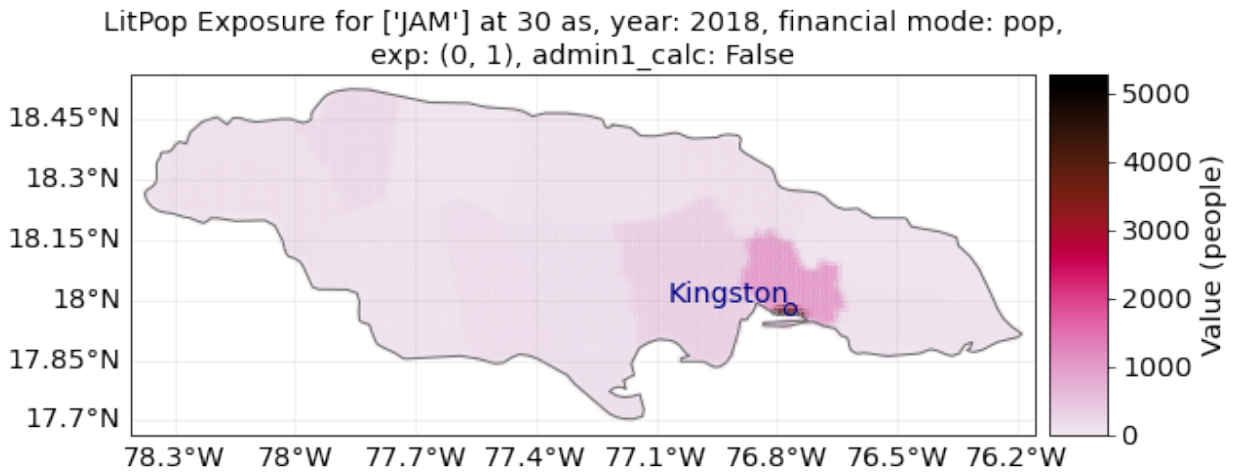
These wrapper methods can be used to produce exposures that are showing purely nightlight intensity or purely population count.

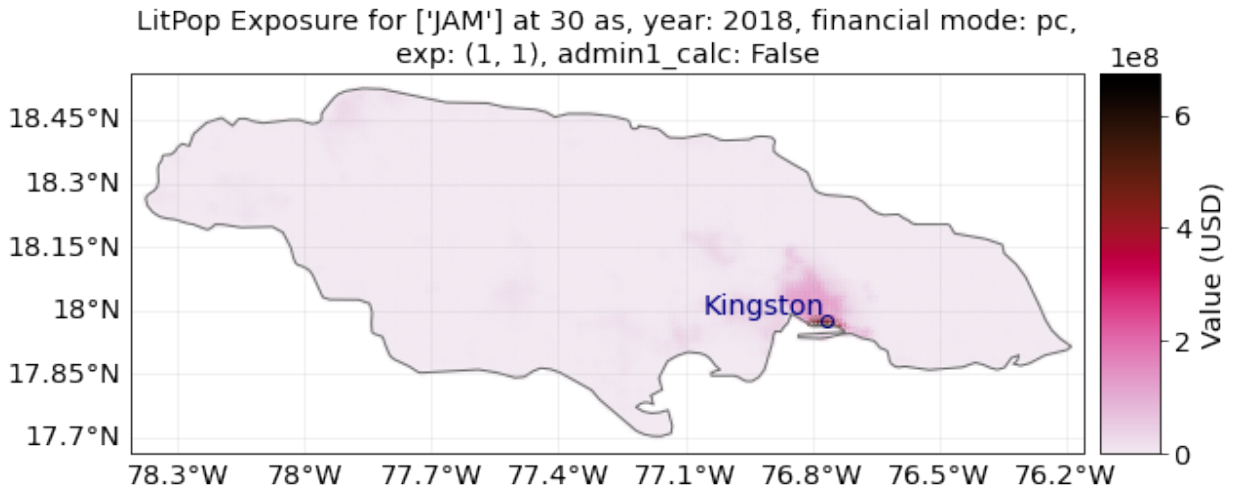
```
res = 30  # If you don't get an output after a very long time with country = "MEX",␣
↪try with res = 100
country = "JAM"  # Try different countries, i.e. 'JAM', 'CHE', 'RWA', 'MEX'
markersize = 4  # for plotting
buffer_deg = 0.04

exp_nightlights = LitPop.from_nightlight_intensity(
    countries=country, res_arcsec=res
)  # nightlight intensity
exp_nightlights.plot_hexbin(linewidth=markersize, buffer=buffer_deg)
# Compare to the population map:
exp_population = LitPop().from_population(countries=country, res_arcsec=res)
exp_population.plot_hexbin(linewidth=markersize, buffer=buffer_deg)
# Compare to default LitPop exposures:
exp = LitPop.from_countries(countries=country, res_arcsec=res)
exp.plot_hexbin(linewidth=markersize, buffer=buffer_deg);
```

```
2021-10-19 17:03:34,705 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2021-10-19 17:03:35,308 - climada.entity.exposures.litpop.litpop - WARNING - Note:␣
↪set_nightlight_intensity sets values to raw nightlight intensity, not to USD. To␣
↪disaggregate asset value proportionally to nightlights^m, call from_countries or␣
↪from_shape with exponents=(m,0).
2021-10-19 17:03:39,867 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2021-10-19 17:03:44,032 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

```
<GeoAxesSubplot:title={'center':"LitPop Exposure for ['JAM'] at 30 as, year: 2018,␣
↪financial mode: pc,\nexp: (1, 1), admin1_calc: False"}>
```







For **Switzerland**, population is resolved on the 3rd administrative level, with 2538 distinct geographical units. Therefore, the purely population-based map is highly resolved.

For **Jamaica**, population is only resolved on the 1st administrative level, with only 14 distinct geographical units. Therefore, the purely population-based map shows large monotonous patches. The combination of Lit and Pop results in a concentration of asset value estimates around the capital city Kingston.

Init LitPop-Exposure from custom shapes

The methods `LitPop.from_shape_and_countries` and `LitPop.from_shape` initiate a LitPop-exposure instance for a given custom shape instead of a country. This can be used to initiate exposure for admin1-regions, i.e. cantons, states, districts, - but also for bounding boxes etc.

The difference between the two methods is that for `from_shape_and_countries`, the exposure for one or more whole countries is initiated first and then it is cropped to the shape. Please make sure that the shape is contained in the given countries. With `from_shape`, the shape is initiated directly which is much more resource efficient but requires a `total_value` to be provided by the user.

A population exposure for a custom shape can be initiated directly via `from_population` without providing `total_value`.

Example: State of Florida

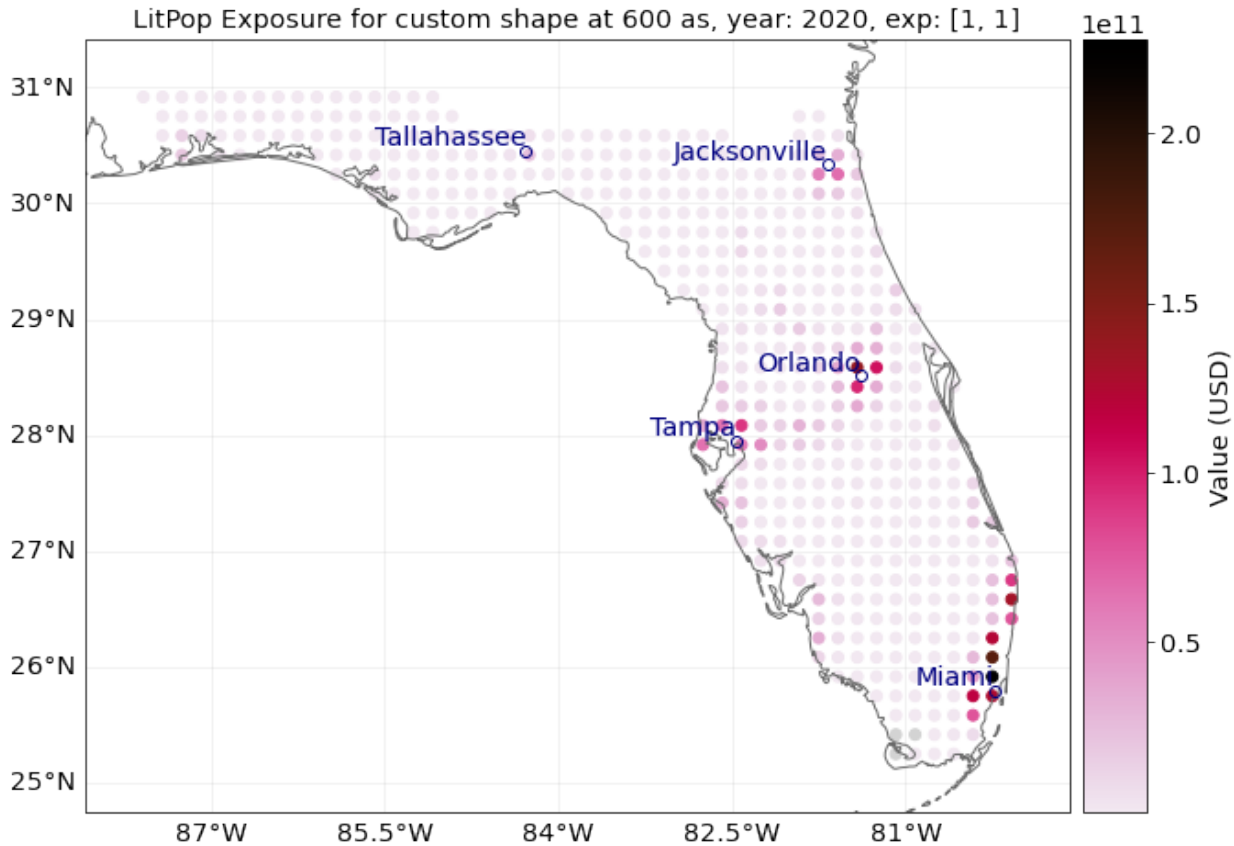
Using `LitPop.from_shape_and_countries` and `LitPop.from_shape` we initiate LitPop exposures for Florida:

```
import time
import climada.util.coordinates as u_coord
import climada.entity.exposures.litpop as lp

country_iso3a = "USA"
state_name = "Florida"
reslution_arcsec = 600
"""First, we need to get the shape of Florida:"""
admin1_info, admin1_shapes = u_coord.get_admin1_info(country_iso3a)
admin1_info = admin1_info[country_iso3a]
admin1_shapes = admin1_shapes[country_iso3a]
admin1_names = [record["name"] for record in admin1_info]
print(admin1_names)
for idx, name in enumerate(admin1_names):
    if admin1_names[idx] == state_name:
        break
print("Florida index: " + str(idx))

"""Secondly, we estimate the `total_value`"""
# `total_value` required user input for `from_shape`, here we assume 5% of total␣
↪value of the whole USA:
total_value = 0.05 * lp._get_total_value_per_country(country_iso3a, "pc", 2020)

"""Then, we can initiate the exposures for Florida:"""
start = time.process_time()
exp = LitPop.from_shape(
    admin1_shapes[idx], total_value, res_arcsec=600, reference_year=2020
)
print(f"\n Runtime `from_shape` : {time.process_time() - start:1.2f} sec.\n")
exp.plot_scatter(vmin=100, buffer=0.5);
```

```
['Minnesota', 'Washington', 'Idaho', 'Montana', 'North Dakota', 'Michigan', 'Maine',
↪'Ohio', 'New Hampshire', 'New York', 'Vermont', 'Pennsylvania', 'Arizona',
↪'California', 'New Mexico', 'Texas', 'Alaska', 'Louisiana', 'Mississippi', 'Alabama
↪', 'Florida', 'Georgia', 'South Carolina', 'North Carolina', 'Virginia', 'District␣
↪of Columbia', 'Maryland', 'Delaware', 'New Jersey', 'Connecticut', 'Rhode Island',
↪'Massachusetts', 'Oregon', 'Hawaii', 'Utah', 'Wyoming', 'Nevada', 'Colorado',
↪'South Dakota', 'Nebraska', 'Kansas', 'Oklahoma', 'Iowa', 'Missouri', 'Wisconsin',
↪'Illinois', 'Kentucky', 'Arkansas', 'Tennessee', 'West Virginia', 'Indiana']
Florida index: 20
```

```
 Runtime `from_shape` : 9.01 sec.
```

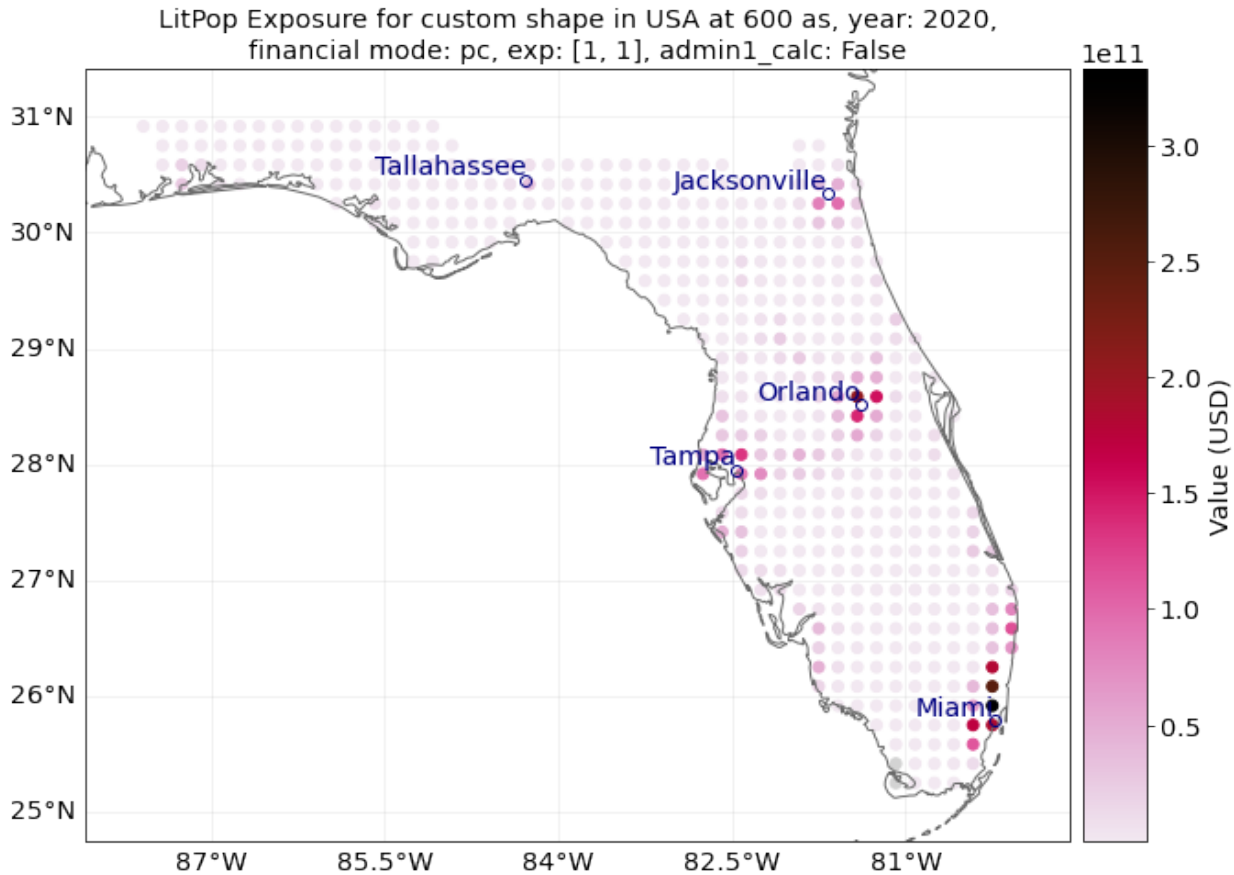
```
<GeoAxesSubplot:title={'center':'LitPop Exposure for custom shape at 600 as, year:␣
↪2020, exp: [1, 1]'}>
```




```
# `from_shape_and_countries` does not require `total_value`, but is slower to compute␣
↪than `from_shape`,
# because first, the exposure for the whole USA is initiated:
start = time.process_time()
exp = LitPop.from_shape_and_countries(
    admin1_shapes[idx], country_iso3a, res_arcsec=600, reference_year=2020
)
print(
    f"\n Runtime `from_shape_and_countries` : {time.process_time() - start:1.2f} sec.\
↪n"
)
exp.plot_scatter(vmin=100, buffer=0.5)
"""Note the differences in computational speed and total value between the two␣
↪approaches"""
```

```
 Runtime `from_shape_and_countries` : 24.49 sec.
```

```
'Note the differences in computational speed and total value between the two␣
↪approaches'
```



Example: Zurich city area

You can also define your own shape as a Polygon:

```
import time
from shapely.geometry import Polygon

"""initiate LitPop exposures for a geographical box around the city of Zurich:"""
bounds = (8.41, 47.25, 8.70, 47.47)  # (min_lon, max_lon, min_lat, max_lat)
total_value = 1000  # required user input for `from_shape`, here we just assume USD␣
↪1000 of total value
shape = Polygon(
    [
        (bounds[0], bounds[3]),
        (bounds[2], bounds[3]),
        (bounds[2], bounds[1]),
        (bounds[0], bounds[1]),
    ]
)
import time

start = time.process_time()
exp = LitPop.from_shape(shape, total_value)
print(f"\n Runtime `from_shape` : {time.process_time() - start:1.2f} sec.\n")
```

(continues on next page)

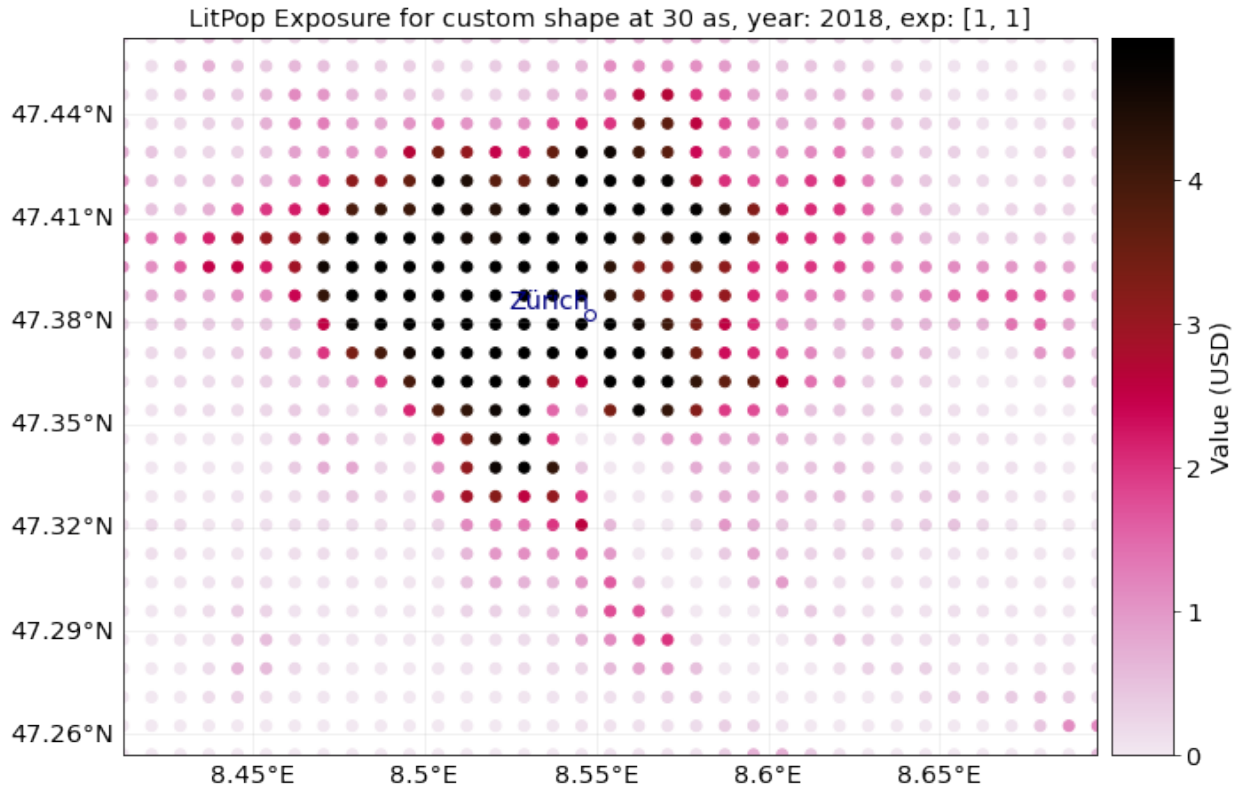
(continued from previous page)

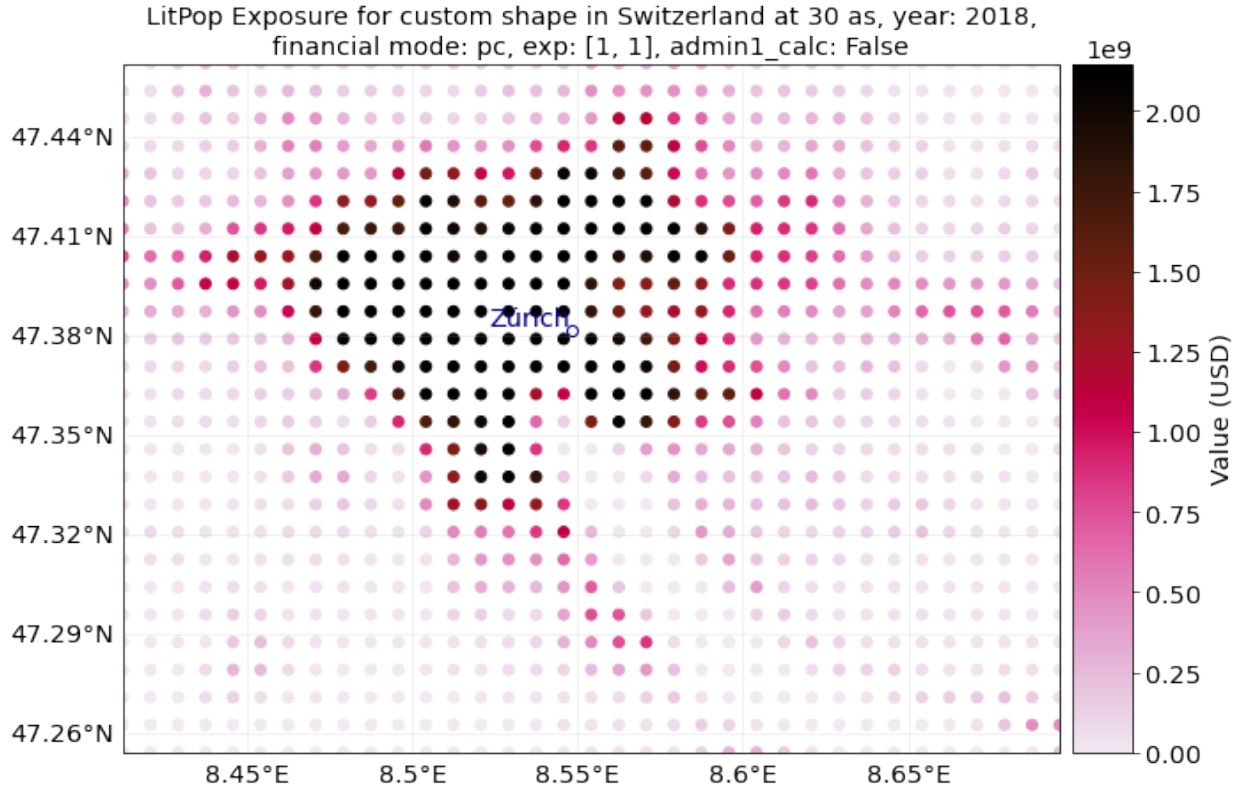
```
exp.plot_scatter()
# `from_shape_and_countries` does not require `total_value`, but is slower to compute:
start = time.process_time()
exp = LitPop.from_shape_and_countries(shape, "Switzerland")
print(
    f"\n Runtime `from_shape_and_countries` : {time.process_time() - start:1.2f} sec.\
↪n"
)
exp.plot_scatter()
"""Note the difference in total value between the two exposure sets!"""

"""For comparison, initiate population exposure for a geographical box around the␣
↪city of Zurich:"""
start = time.process_time()
exp_pop = LitPop.from_population(shape=shape)
print(f"\n Runtime `from_population` : {time.process_time() - start:1.2f} sec.\n")
exp_pop.plot_scatter()

"""Population exposure for a custom shape can be initiated directly via `set_
↪population` without providing `total_value`"""
```

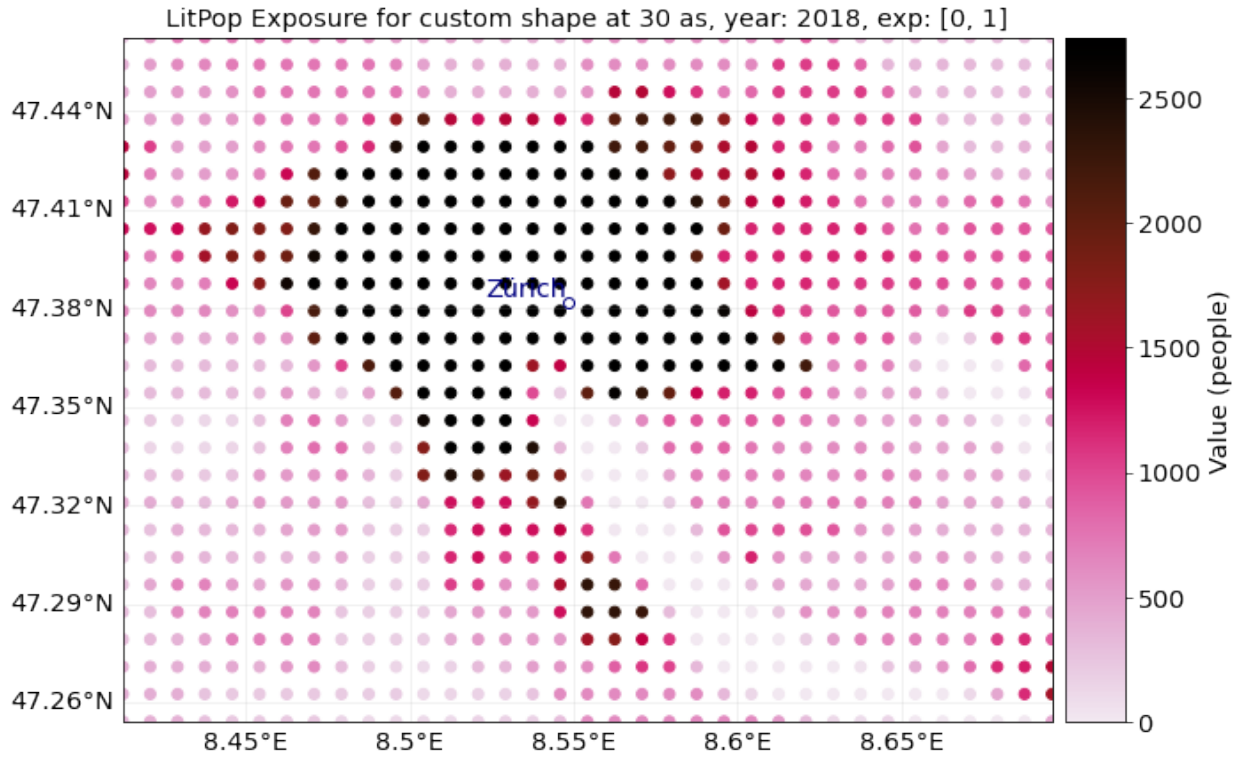
```
 Runtime `from_shape` : 0.51 sec.

2021-10-19 17:04:24,606 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020

 Runtime `from_shape_and_countries` : 3.18 sec.
```

```
 Runtime `from_population` : 0.75 sec.
```

```
'Population exposure for a custom shape can be initiated directly via `set_
↪population` without providing `total_value`'
```


LitPop Exposure for custom shape at 30 as, year: 2018, exp: [1, 1]
47.44°N
47.41°N
47.38°N
47.35°N
47.32°N
47.29°N
47.26°N
8.45°E
8.5°E
8.55°E
8.6°E
8.65°E
Zürich
Value (USD)
0
1
2
3
4


LitPop Exposure for custom shape in Switzerland at 30 as, year: 2018,
financial mode: pc, exp: [1, 1], admin1_calc: False
1e9
47.44°N
47.41°N
47.38°N
47.35°N
47.32°N
47.29°N
47.26°N
8.45°E
8.5°E
8.55°E
8.6°E
8.65°E
Zürich
Value (USD)
0.00
0.25
0.50
0.75
1.00
1.25
1.50
1.75
2.00



Sub-national (admin-1) GDP as intermediate downscaling layer

In order to improve downscaling for countries with large regional differences within, a subnational breakdown of GDP can be used as an intermediate downscaling layer wherever available.

The sub-national (admin-1) GDP-breakdown needs to be added manually as a ".xls"-file to the folder `data/system/GSDP` in the CLIMADA-directory. Currently, such data is provided for more than 10 countries, including USA, India, and China.

The xls-file requires at least the following columns (with names specified in row 1):

- `State_Province`: Names of admin-1 regions, i.e. states, cantons, provinces. Names need to match the naming of admin-1 shapes in the data used by the python package `cartopy.io` (c.f. `shapereader.natural_earth(name='admin_1_states_provinces')`)
- `GSDP_ref`: value of sub-national GDP to be used (absolute or relative values)
- `Postal`, optional: Alternative identifier of region, if names do not match with cartopy. Needs to correspond to the Postal-identifiers used in the shapereader of `cartopy.io`.

Please note that while admin1-GDP will per definition improve the downscaling of *GDP*, it might not necessarily improve the downscaling quality for other asset bases like produced capital (pc).

How To:

The intermediate downscaling layer can be activated with the parameter `admin1_calc=True`.

```
# Initiate GDP-Entity for Switzerland, with and without admin1_calc:

ent_adm0 = LitPop.from_countries(
    "CHE", res_arcsec=120, fin_mode="gdp", admin1_calc=False
```

(continues on next page)

(continued from previous page)

```
)
ent_adm0.check()

ent_adm1 = LitPop.from_countries(
    "CHE", res_arcsec=120, fin_mode="gdp", admin1_calc=True
)
ent_adm1.check()

print("Done.")
```

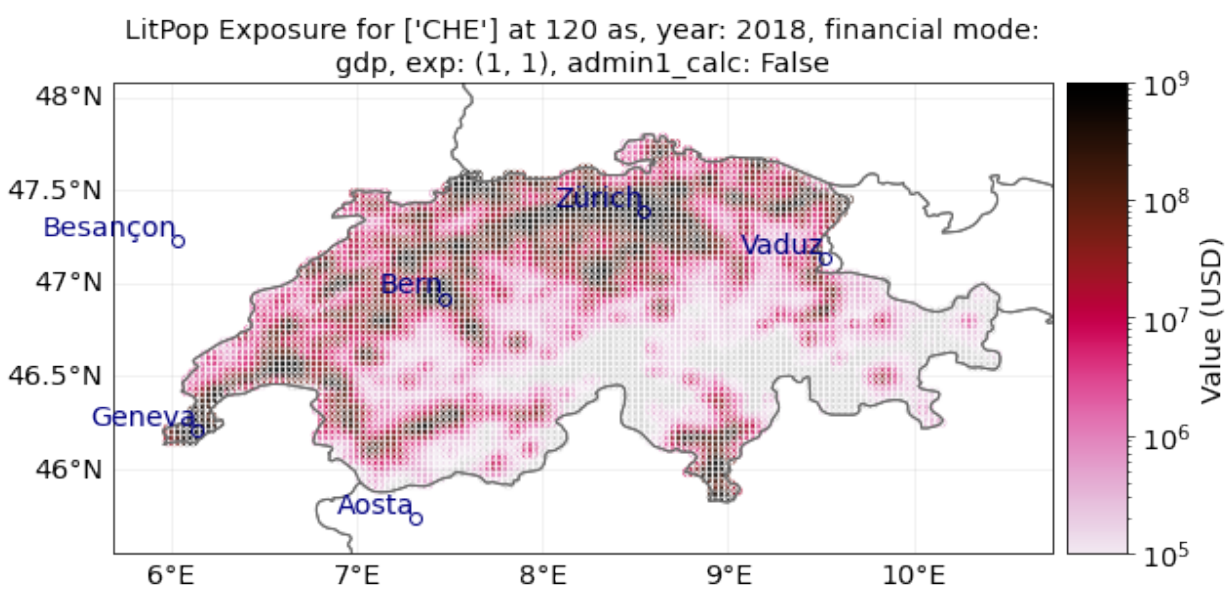
```
2021-10-19 17:04:31,363 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

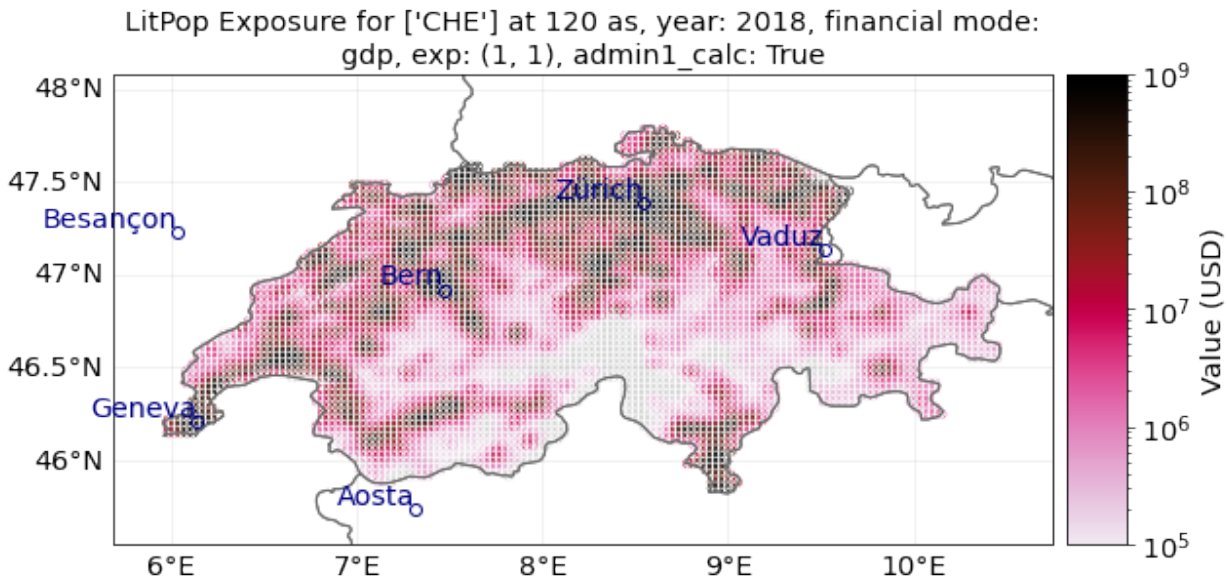
```
Done.
```

```
# Plotting:
from matplotlib import colors

norm = colors.LogNorm(vmin=1e5, vmax=1e9)  # setting range for the log-normal scale
markersize = 5
ent_adm0.plot_hexbin(buffer=0.3, norm=norm, linewidth=markersize)
ent_adm1.plot_hexbin(buffer=0.3, norm=norm, linewidth=markersize)

print("admin-0: First figure")
print("admin-1: Second figure")
"""Do you spot the small differences in Graubünden (eastern Switzerland)?"""
```

```
admin-0: First figure
admin-1: Second figure
```

```
'Do you spot the small differences in Graubünden (eastern Switzerland)?'
```





2.4.3 How to use polygons or lines as exposure

Introduction

Exposure in CLIMADA are usually represented as individual points or a raster of points. See *Exposures* tutorial to learn how to fill and use exposures. In this tutorial we show you how to run the impact calculation chain if you have polygons or lines to start with. The approach provides an all-in-one method for impact calculation: `calc_geom_impact` It features three sub-steps, for which the current util module `lines_polys_handler` also provides separate functions:

1. Disaggregation of line and polygon data into point exposure:
   - Interpolate geometries to points to fit in an `Exposure` instance;
   - Disaggregate the respective geometry values to the point values
2. Perform the impact calculation in CLIMADA with the point exposure
3. Aggregate the calculated point `Impact` back to an impact instance for the initial polygons or lines

Note: Polygons or lines can be useful to represent specific types of exposures such as infrastructures (e.g. roads) or landuse types (e.g. crops, forests). In CLIMADA, it is possible to retrieve such specific exposure types using OpenStreetMap data. Please refer to the associated tutorial to learn how to do so.

Quick example

Get example polygons (provinces), lines (rails), points exposure for the Netherlands, and create one single Exposures. Get demo winter storm hazard and a corresponding impact function.

```
from climada.util.api_client import Client
import climada.util.lines_polys_handler as u_lp
from climada.entity.impact_funcs import ImpactFuncSet
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity import Exposures

HAZ = Client().get_hazard("storm_europe", name="test_haz_WS_nl", status="test_dataset
↪")
```

(continues on next page)

(continued from previous page)

```
EXP_POLY = Client().get_exposures(
    "base", name="test_polygon_exp", status="test_dataset"
)
EXP_LINE = Client().get_exposures("base", name="test_line_exp", status="test_dataset")
EXP_POINT = Client().get_exposures("base", name="test_point_exp", status="test_dataset
↪")

EXP_MIX = Exposures.concat([EXP_POLY, EXP_LINE, EXP_POINT])

IMPF = ImpfStormEurope.from_welker()
IMPF_SET = ImpactFuncSet([IMPF])
```

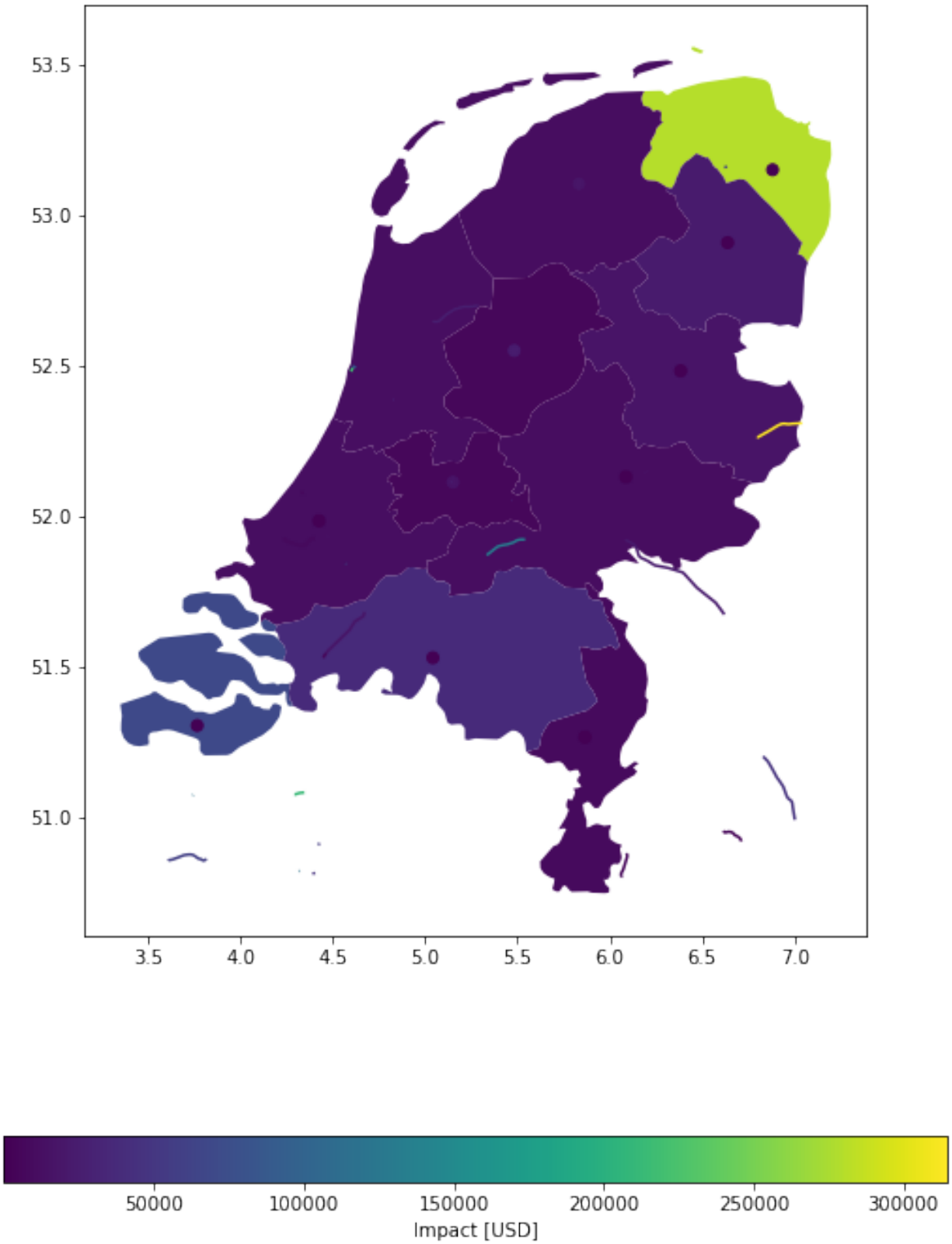
Compute the impact in one line.

```
# disaggregate in the same CRS as the exposures are defined (here degrees),␣
↪resolution 1degree
# divide values on points
# aggregate by summing

impact = u_lp.calc_geom_impact(
    exp=EXP_MIX,
    impf_set=IMPF_SET,
    haz=HAZ,
    res=0.2,
    to_meters=False,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
    agg_met=u_lp.AggMethod.SUM,
)
```

```
2022-06-24 13:16:15,277 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:15,284 - climada.entity.exposures.base - INFO - Matching 183␣
↪exposures with 9944 centroids.
2022-06-24 13:16:15,285 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:15,295 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:15,296 - climada.engine.impact - INFO - Calculating damage for 182␣
↪assets (>0) and 2 events.
```

```
/Users/ckropf/Documents/Climada/climada_python/climada/util/lines_polys_handler.
↪py:931: UserWarning: Geometry is in a geographic CRS. Results from 'length' are␣
↪likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected␣
↪CRS before this operation.

  line_lengths = gdf_lines.length
```

```
u_lp.plot_eai_exp_geom(impact);
```




```
# disaggregate in meters
```

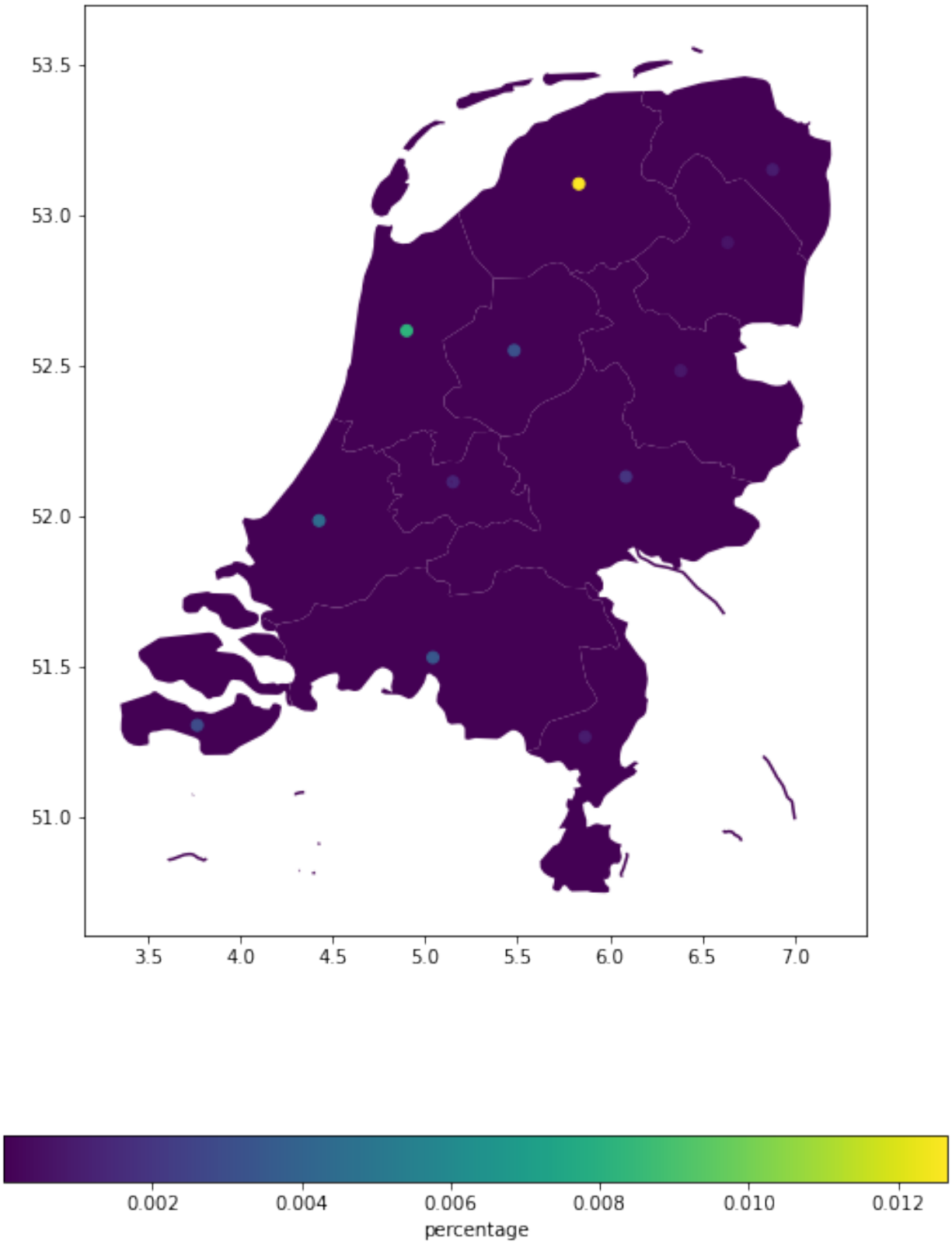
(continues on next page)

(continued from previous page)

```
# same value for each point, fixed to 1 (allows to get percentages of affected␣
↪surface/distance)
# aggregate by summing

impact = u_lp.calc_geom_impact(
    exp=EXP_MIX,
    impf_set=IMPF_SET,
    haz=HAZ,
    res=1000,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.FIX,
    disagg_val=1.0,
    agg_met=u_lp.AggMethod.SUM,
);
```

```
2022-06-24 13:16:17,387 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:18,069 - climada.entity.exposures.base - INFO - Matching 37357␣
↪exposures with 9944 centroids.
2022-06-24 13:16:18,073 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:18,114 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:18,117 - climada.engine.impact - INFO - Calculating damage for 37357␣
↪assets (>0) and 2 events.
```

```
import matplotlib.pyplot as plt

ax = u_lp.plot_eai_exp_geom(
    impact, legend_kwds={"label": "percentage", "orientation": "horizontal"}
)
```


53.5
53.0
52.5
52.0
51.5
51.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
0.002
0.004
0.006
0.008
0.010
0.012
percentage

Polygons

Polygons or shapes are a common geographical representation of countries, states etc. as for example in NaturalEarth. Map data, as for example buildings, etc. obtained from openstreetmap (see tutorial here), also frequently come as (multi-)polygons. Here we want to show you how to deal with exposure information as polygons.

Load data

Lets assume we have the following data given. The polygons of the admin-1 regions of the Netherlands and an exposure value each, which we gather in a geodataframe. We want to know the Impact of Lothar on each admin-1 region.

In this tutorial, we shall see how to compute impacts for exposures defined on shapely geometries (polygons and/or lines). The basic principle is to disaggregate the geometries to a raster of points, compute the impact per points, and then re-aggregate. To do so, several methods are available. Here is a brief overview.

```
# Imports
import geopandas as gpd
import pandas as pd
from pathlib import Path

from climada.entity import Exposures
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity.impact_funcs import ImpactFuncSet
from climada.engine import ImpactCalc
from climada.hazard.storm_europe import StormEurope
import climada.util.lines_polys_handler as u_lp
from climada.util.constants import DEMO_DIR, WS_DEMO_NC
```

```
def gdf_poly():
    from cartopy.io import shapereader
    from climada_petals.entity.exposures.black_marble import country_iso_geom

    # open the file containing the Netherlands admin-1 polygons
    shp_file = shapereader.natural_earth(
        resolution="10m", category="cultural", name="admin_0_countries"
    )
    shp_file = shapereader.Reader(shp_file)

    # extract the NL polygons
    prov_names = {
        "Netherlands": [
            "Groningen",
            "Drenthe",
            "Overijssel",
            "Gelderland",
            "Limburg",
            "Zeeland",
            "Noord-Brabant",
            "Zuid-Holland",
            "Noord-Holland",
            "Friesland",
            "Flevoland",
            "Utrecht",
        ]
```

(continues on next page)

(continued from previous page)

```
    }
    polygon_Netherlands, polygons_prov_NL = country_iso_geom(prov_names, shp_file)
    prov_geom_NL = {
        prov: geom
        for prov, geom in zip(
            list(prov_names.values())[0], list(polygons_prov_NL.values())[0]
        )
    }

    # assign a value to each admin-1 area (assumption 100'000 USD per inhabitant)
    population_prov_NL = {
        "Drenthe": 493449,
        "Flevoland": 422202,
        "Friesland": 649988,
        "Gelderland": 2084478,
        "Groningen": 585881,
        "Limburg": 1118223,
        "Noord-Brabant": 2562566,
        "Noord-Holland": 2877909,
        "Overijssel": 1162215,
        "Zuid-Holland": 3705625,
        "Utrecht": 1353596,
        "Zeeland": 383689,
    }
    value_prov_NL = {
        n: 100000 * population_prov_NL[n] for n in population_prov_NL.keys()
    }

    # combine into GeoDataFrame and add a coordinate reference system to it:
    df1 = pd.DataFrame.from_dict(
        population_prov_NL, orient="index", columns=["population"]
    ).join(pd.DataFrame.from_dict(value_prov_NL, orient="index", columns=["value"]))
    df1["geometry"] = [prov_geom_NL[prov] for prov in df1.index]
    gdf_polys = gpd.GeoDataFrame(df1)
    gdf_polys = gdf_polys.set_crs(epsg=4326)
    return gdf_polys
```

```
exp_nl_poly = Exposures(gdf_poly())
exp_nl_poly.gdf["impf_WS"] = 1
exp_nl_poly.gdf.head()
```

```
            population         value  \
Drenthe         493449   49344900000
Flevoland       422202   42220200000
Friesland       649988   64998800000
Gelderland     2084478  208447800000
Groningen       585881   58588100000

                                                  geometry  impf_WS
Drenthe     POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
Flevoland   POLYGON ((5.74046 52.83874, 5.75012 52.83507, ...        1
```

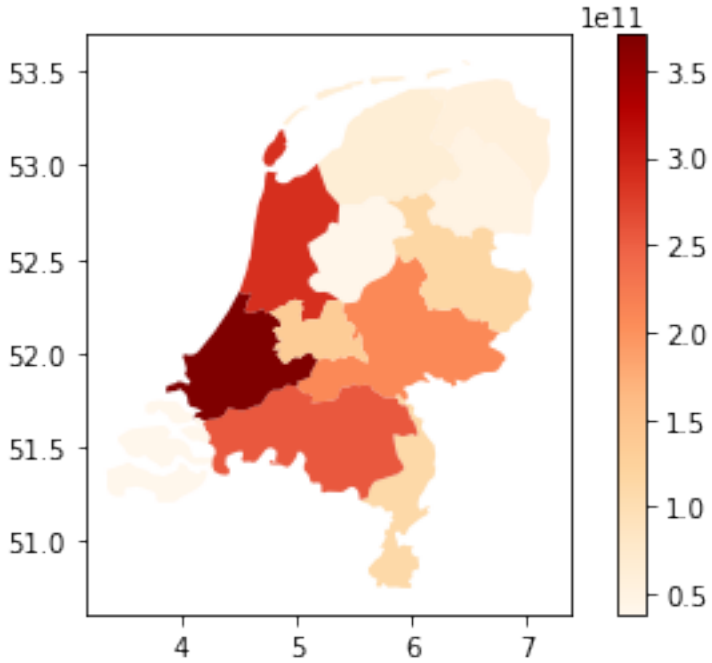
(continues on next page)

(continued from previous page)

```
Friesland   MULTIPOLYGON (((5.17977 53.00899, 5.27947 53.0...        1
Gelderland  POLYGON ((6.77158 52.10879, 6.76587 52.10840, ...        1
Groningen   MULTIPOLYGON (((7.19459 53.24502, 7.19747 53.2...        1
```

```
# take a look
exp_nl_poly.gdf.plot("value", legend=True, cmap="OrRd")
```

```
<AxesSubplot:>
```




```
# define hazard
storms = StormEurope.from_footprints(WS_DEMO_NC)
# define impact function
impf = ImpfStormEurope.from_welker()
impf_set = ImpactFuncSet([impf])
```

```
2022-06-24 13:16:20,039 - climada.hazard.storm_europe - INFO - Constructing centroids␣
↪from /Users/ckropf/climada/demo/data/fp_lothar_crop-test.nc
2022-06-24 13:16:20,124 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
```

```
/Users/ckropf/Documents/Climada/climada_python/climada/hazard/centroids/centr.py:822:␣
↪UserWarning: Geometry is in a geographic CRS. Results from 'buffer' are likely␣
↪incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected CRS␣
↪before this operation.

  xy_pixels = self.geometry.buffer(res / 2).envelope
```

```
2022-06-24 13:16:22,004 - climada.hazard.storm_europe - INFO - Commencing to iterate␣
↪over netCDF files.
```

Compute polygon impacts - all in one

All in one: The main method `calc_geom_impact` provides several disaggregation keywords, specifiying

- the target resolution (`res`),
- the method on how to distribute the values of the original geometries onto the newly generated interpolated points (`disagg_met`)
- the source (and number) of the value to be distributed (`disagg_val`).
- the aggregation method (`agg_met`)

`disagg_met` can be either fixed (`FIX`), replicating the original shape's value onto all points, or divided evenly (`DIV`), in which case the value is divided equally onto all new points. `disagg_val` can either be taken directly from the exposure gdf's value column (`None`) or be indicated here explicitly (`float`). Resolution can be given in the gdf's original (mostly degrees lat/lon) format, or in metres. `agg_met` can currently be only (`SUM`) were the value is summed over all points in the geometry.

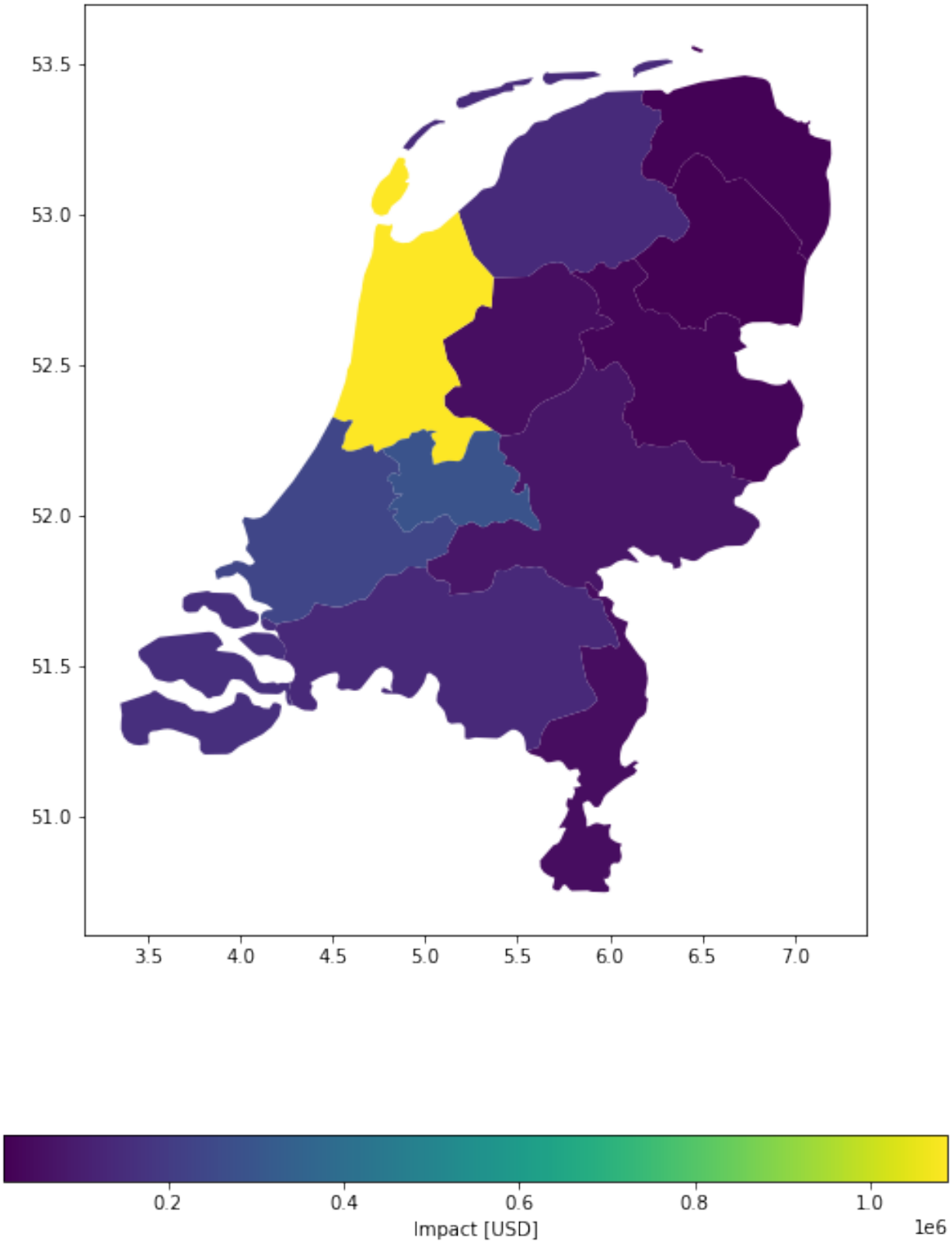
Polygons can also be disaggregated on a given fixed grid, see example below

Example 1: Target resolution in degrees lat/lon, equal (average) distribution of values from exposure gdf among points.

```
imp_deg = u_lp.calc_geom_impact(
    exp=exp_nl_poly,
    impf_set=impf_set,
    haz=storms,
    res=0.005,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
    agg_met=u_lp.AggMethod.SUM,
)
```

```
2022-06-24 13:16:30,535 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:33,946 - climada.entity.exposures.base - INFO - Matching 195323␣
↪exposures with 9944 centroids.
2022-06-24 13:16:33,950 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:34,140 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:34,144 - climada.engine.impact - INFO - Calculating damage for␣
↪195323 assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_deg);
```

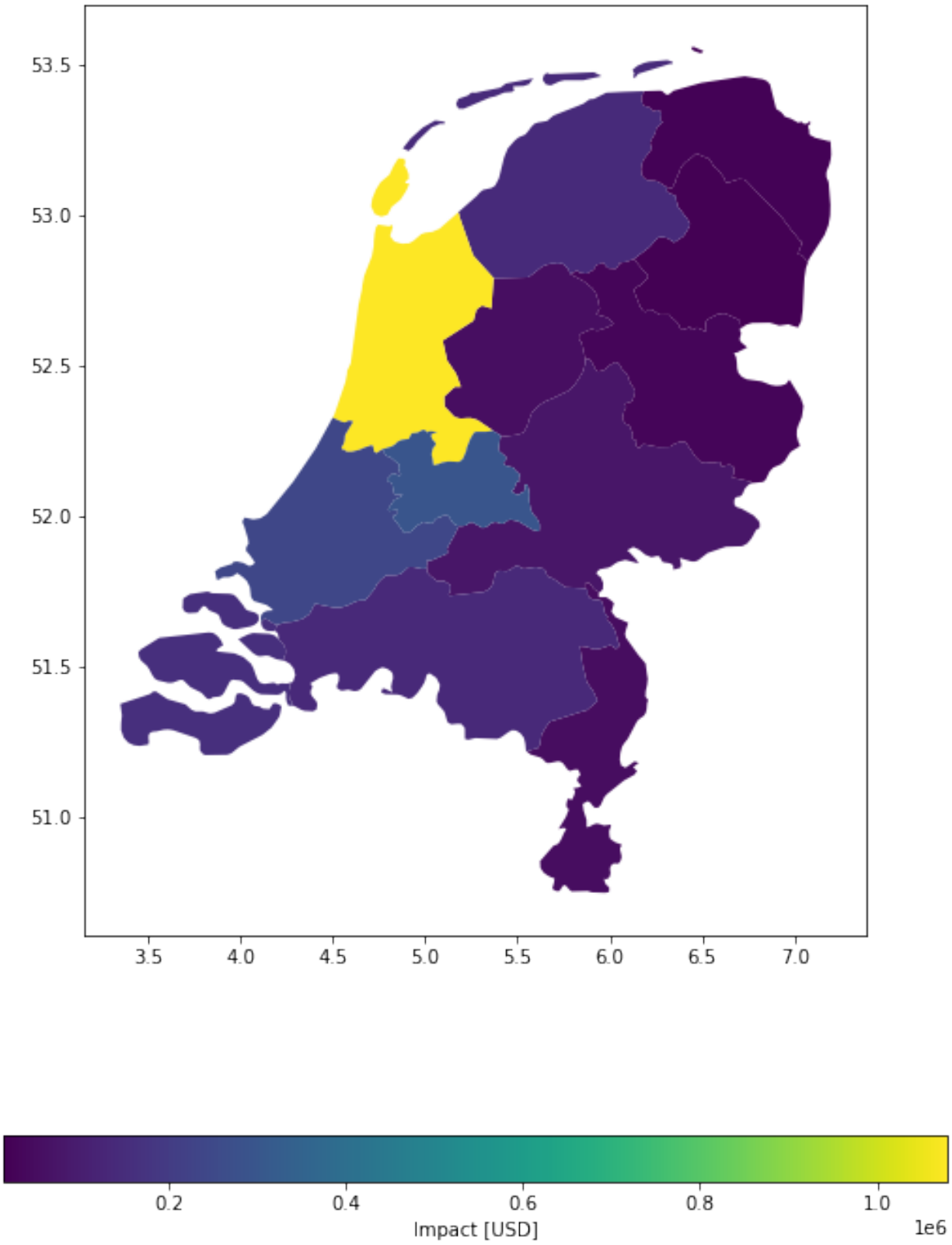


Example 2: Target resolution in metres, equal (divide) distribution of values from exposure gdf among points.

```
imp_m = u_lp.calc_geom_impact(
    exp=exp_nl_poly,
    impf_set=impf_set,
    haz=storms,
    res=500,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
    agg_met=u_lp.AggMethod.SUM,
)
```

```
2022-06-24 13:16:41,366 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:44,164 - climada.entity.exposures.base - INFO - Matching 148369␣
↪exposures with 9944 centroids.
2022-06-24 13:16:44,168 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:44,387 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:44,390 - climada.engine.impact - INFO - Calculating damage for␣
↪148369 assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_m);
```



For this specific case, both disaggregation methods provide a relatively similar result, given the chosen numbers:

```
(imp_deg.eai_exp - imp_m.eai_exp) / imp_deg.eai_exp
```

```
array([ 0.00614586, -0.01079377,  0.00021355,  0.00140608,  0.0038771 ,
       -0.0066888 , -0.00171755,  0.00741871, -0.00107029,  0.00424221,
       -0.01838225,  0.01811858])
```

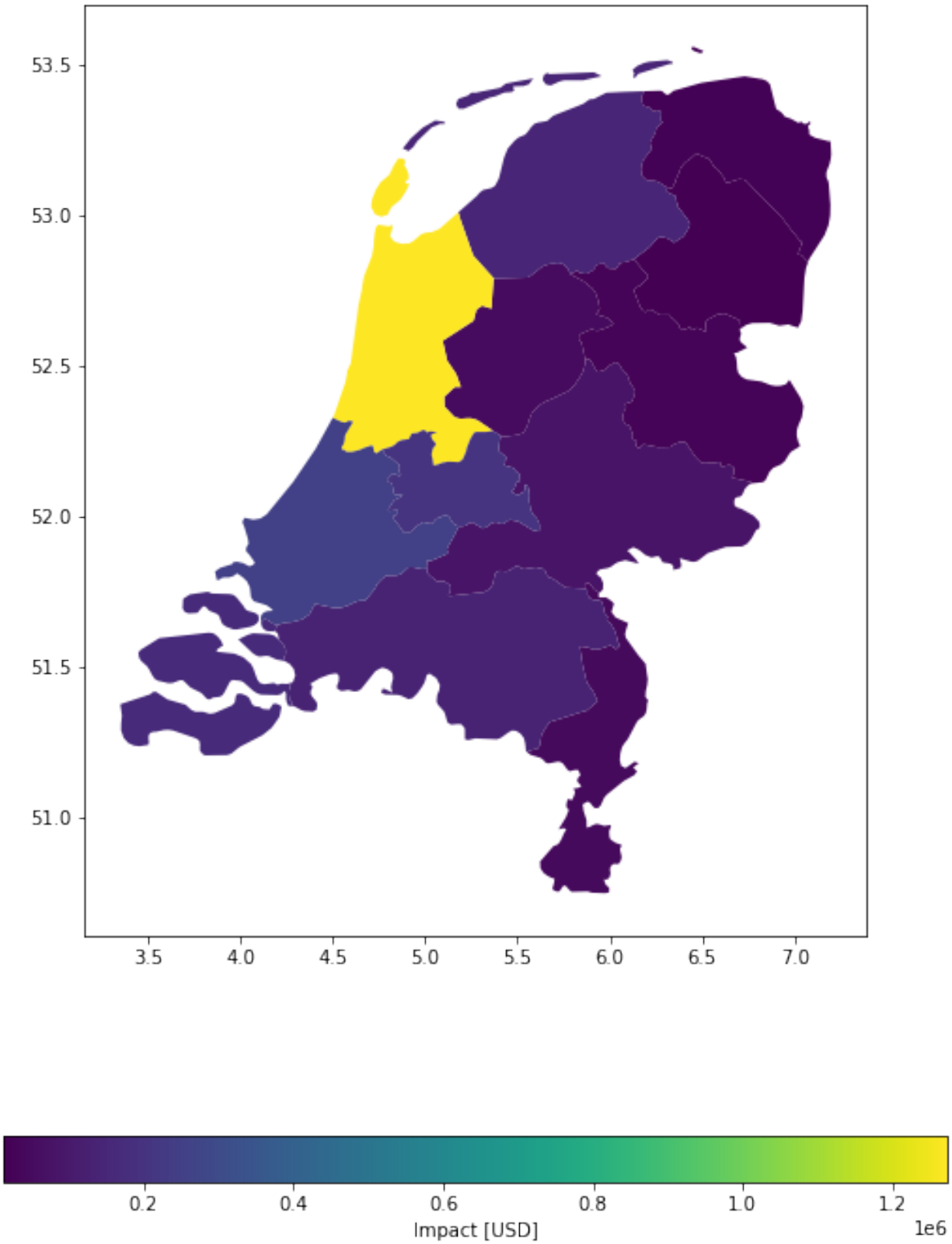
Example 3: Target predefined grid, equal (divide) distribution of values from exposure gdf among points.

```
# regular grid from exposures bounds
import climada.util.coordinates as u_coord

res = 0.1
(_, _, xmax, ymax) = exp_nl_poly.gdf.geometry.bounds.max()
(xmin, ymin, _, _) = exp_nl_poly.gdf.geometry.bounds.min()
bounds = (xmin, ymin, xmax, ymax)
height, width, trafo = u_coord.pts_to_raster_meta(bounds, (res, res))
x_grid, y_grid = u_coord.raster_to_meshgrid(trafo, width, height)
```

```
imp_g = u_lp.calc_grid_impact(
    exp=exp_nl_poly,
    impf_set=impf_set,
    haz=storms,
    grid=(x_grid, y_grid),
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
    agg_met=u_lp.AggMethod.SUM,
)
```

```
2022-06-24 13:16:45,086 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:16:45,100 - climada.entity.exposures.base - INFO - Matching 486␣
↪exposures with 9944 centroids.
2022-06-24 13:16:45,102 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:45,110 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:45,111 - climada.engine.impact - INFO - Calculating damage for 486␣
↪assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_g);
```


53.5
53.0
52.5
52.0
51.5
51.0
3.5
4.0
4.5
5.0
5.5
6.0
6.5
7.0
0.2
0.4
0.6
0.8
1.0
1.2
Impact [USD]
1e6

Compute polygon impacts - step by step

Step 1: Disaggregate polygon exposures to points. It is useful to do this separately, when the discretized exposure is used several times, for example, to compute with different hazards.

Several disaggregation methods can be used as shown below:

```
# Disaggregate exposure to 10'000 metre grid, each point gets average value within␣
↪polygon.
exp_pnt = u_lp.exp_geom_to_pnt(
    exp_nl_poly,
    res=10000,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
)
exp_pnt.gdf.head()
```

```
2022-06-24 13:16:45,445 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
           population         value  \
Drenthe 0      493449  2.056038e+09
        1      493449  2.056038e+09
        2      493449  2.056038e+09
        3      493449  2.056038e+09
        4      493449  2.056038e+09

                                                geometry_orig  impf_WS  \
Drenthe 0  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        1  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        2  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        3  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        4  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1

                           geometry   latitude  longitude
Drenthe 0  POINT (6.21931 52.75939)  52.759394   6.219308
        1  POINT (6.30914 52.75939)  52.759394   6.309139
        2  POINT (6.39897 52.75939)  52.759394   6.398971
        3  POINT (6.48880 52.75939)  52.759394   6.488802
        4  POINT (6.57863 52.75939)  52.759394   6.578634
```

```
# Disaggregate exposure to 0.1° grid, no value disaggregation specified --> replicate␣
↪initial value
exp_pnt2 = u_lp.exp_geom_to_pnt(
    exp_nl_poly,
    res=0.1,
    to_meters=False,
    disagg_met=u_lp.DisaggMethod.FIX,
    disagg_val=None,
)
exp_pnt2.gdf.head()
```

```
2022-06-24 13:16:45,516 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
           population        value  \
Drenthe 0      493449  49344900000
        1      493449  49344900000
        2      493449  49344900000
        3      493449  49344900000
        4      493449  49344900000

                                                 geometry_orig  impf_WS  \
Drenthe 0  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        1  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        2  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        3  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        4  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1

                            geometry   latitude  longitude
Drenthe 0  POINT (6.22948 52.71147)  52.711466   6.229476
        1  POINT (6.32948 52.71147)  52.711466   6.329476
        2  POINT (6.42948 52.71147)  52.711466   6.429476
        3  POINT (6.52948 52.71147)  52.711466   6.529476
        4  POINT (6.62948 52.71147)  52.711466   6.629476
```

```
# Disaggregate exposure to 1'000 metre grid, each point gets value corresponding to
# its representative area (1'000^2).
exp_pnt3 = u_lp.exp_geom_to_pnt(
    exp_nl_poly,
    res=1000,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.FIX,
    disagg_val=10e6,
)
exp_pnt3.gdf.head()
```

```
2022-06-24 13:16:47,258 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
           population       value  \
Drenthe 0      493449  10000000.0
        1      493449  10000000.0
        2      493449  10000000.0
        3      493449  10000000.0
        4      493449  10000000.0

                                                 geometry_orig  impf_WS  \
Drenthe 0  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        1  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        2  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        3  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
        4  POLYGON ((7.07215 52.84132, 7.06198 52.82401, ...        1
```

(continues on next page)

(continued from previous page)

```
                           geometry   latitude  longitude
Drenthe 0  POINT (6.38100 52.62624)  52.626236   6.381005
        1  POINT (6.38999 52.62624)  52.626236   6.389988
        2  POINT (6.39897 52.62624)  52.626236   6.398971
        3  POINT (6.40795 52.62624)  52.626236   6.407954
        4  POINT (6.41694 52.62624)  52.626236   6.416937
```

```
# Disaggregate exposure to 1'000 metre grid, each point gets value corresponding to 1
# After dissagregation, each point has a value equal to the percentage of area of the␣
↪polygon
exp_pnt4 = u_lp.exp_geom_to_pnt(
    exp_nl_poly,
    res=1000,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=1,
)
exp_pnt4.gdf.tail()
```

```
2022-06-24 13:16:49,929 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
              population     value  \
Zeeland 1897      383689  0.000526
        1898      383689  0.000526
        1899      383689  0.000526
        1900      383689  0.000526
        1901      383689  0.000526

                                                 geometry_orig  impf_WS  \
Zeeland 1897  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        1898  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        1899  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        1900  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        1901  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1

                                 geometry   latitude  longitude
Zeeland 1897  POINT (3.92434 51.73813)  51.738126   3.924336
        1898  POINT (3.93332 51.73813)  51.738126   3.933320
        1899  POINT (3.94230 51.73813)  51.738126   3.942303
        1900  POINT (3.95129 51.73813)  51.738126   3.951286
        1901  POINT (3.96027 51.73813)  51.738126   3.960269
```

```
# disaggregate on pre-defined grid
# regular grid from exposures bounds
import climada.util.coordinates as u_coord

res = 0.1
(_, _, xmax, ymax) = exp_nl_poly.gdf.geometry.bounds.max()
(xmin, ymin, _, _) = exp_nl_poly.gdf.geometry.bounds.min()
bounds = (xmin, ymin, xmax, ymax)
```

(continues on next page)

(continued from previous page)

```
height, width, trafo = u_coord.pts_to_raster_meta(bounds, (res, res))
x_grid, y_grid = u_coord.raster_to_meshgrid(trafo, width, height)
exp_pnt5 = u_lp.exp_geom_to_grid(
    exp_nl_poly, grid=(x_grid, y_grid), disagg_met=u_lp.DisaggMethod.DIV, disagg_val=1
)
exp_pnt5.gdf.tail()
```

```
2022-06-24 13:16:50,765 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
            population     value  \
Zeeland 17      383689  0.045455
        18      383689  0.045455
        19      383689  0.045455
        20      383689  0.045455
        21      383689  0.045455

                                                geometry_orig  impf_WS  \
Zeeland 17  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        18  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        19  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        20  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1
        21  MULTIPOLYGON (((3.45388 51.23563, 3.42194 51.2...        1

                            geometry  latitude  longitude
Zeeland 17  POINT (3.84941 51.54755)  51.54755   3.849415
        18  POINT (4.14941 51.54755)  51.54755   4.149415
        19  POINT (3.94941 51.64755)  51.64755   3.949415
        20  POINT (4.04941 51.64755)  51.64755   4.049415
        21  POINT (4.14941 51.64755)  51.64755   4.149415
```

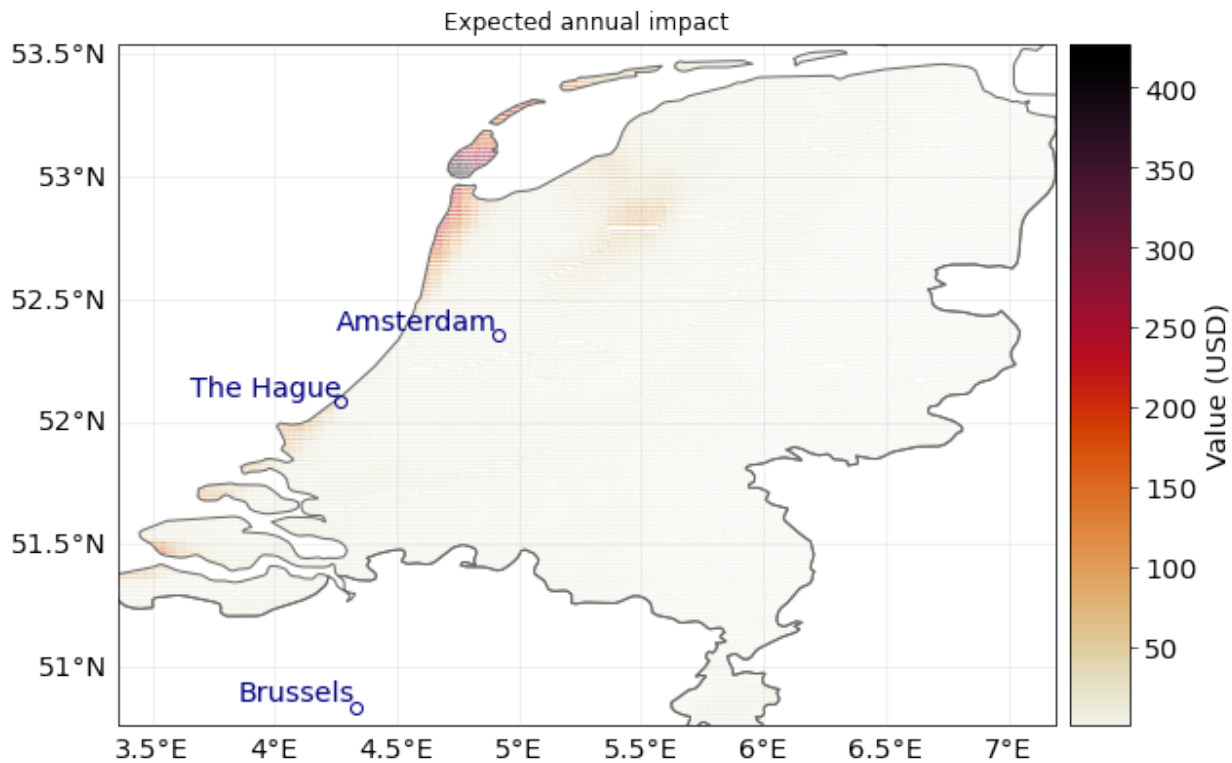
Step 2: Calculate point impacts & re-aggregate them afterwards

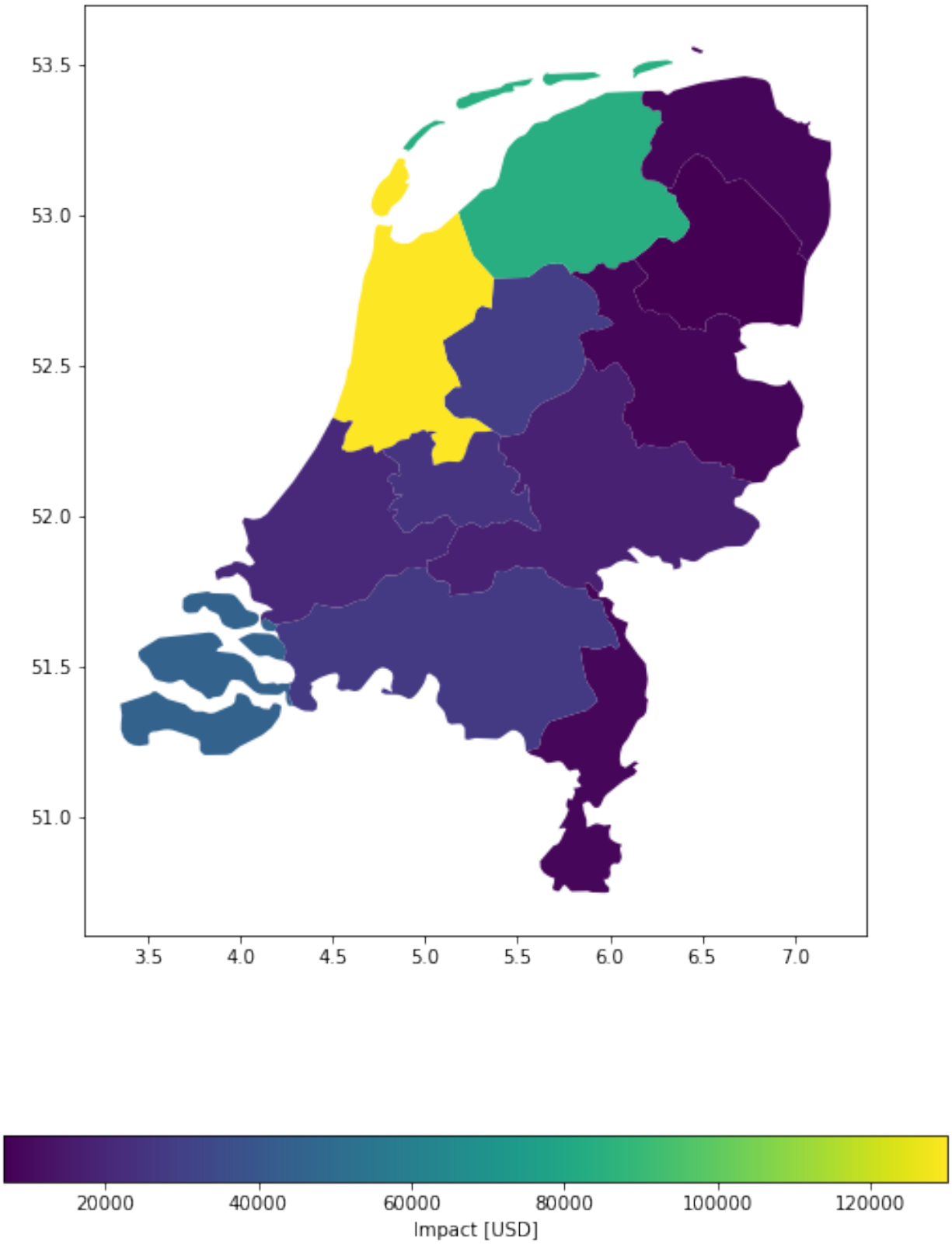
```
# Point-impact
imp_pnt = ImpactCalc(exp_pnt3, impf_set, hazard=storms).impact(save_mat=True)

# Aggregated impact (Note that you need to pass the gdf and not the exposures)
imp_geom = u_lp.impact_pnt_agg(imp_pnt, exp_pnt3.gdf, agg_met=u_lp.AggMethod.SUM)
```

```
2022-06-24 13:16:50,793 - climada.entity.exposures.base - INFO - Matching 37082␣
↪exposures with 9944 centroids.
2022-06-24 13:16:50,796 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:50,929 - climada.engine.impact - INFO - Calculating damage for 37082␣
↪assets (>0) and 2 events.
```

```
# Plot point-impacts and aggregated impacts
imp_pnt.plot_hexbin_eai_exposure()
u_lp.plot_eai_exp_geom(imp_geom);
```

```
2022-06-24 13:16:51,002 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'epsg:4326'. Using projection PlateCarree in plot.
```





Aggregated impact, in detail

```
# aggregate impact matrix
mat_agg = u_lp._aggregate_impact_mat(imp_pnt, exp_pnt3.gdf, agg_met=u_lp.AggMethod.
↪SUM)

eai_exp = u_lp.eai_exp_from_mat(imp_mat=mat_agg, freq=imp_pnt.frequency)
at_event = u_lp.at_event_from_mat(imp_mat=mat_agg)
aai_agg = u_lp.aai_agg_from_at_event(at_event=at_event, freq=imp_pnt.frequency)
```

```
eai_exp
```

```
array([  6753.45186608,  28953.21638482,  84171.28903387,  18014.15630989,
         8561.18507994,   8986.13653385,  27446.46387061, 130145.29903078,
         8362.17243334,  20822.87844894,  25495.46296087,  45121.14833362])
```

```
at_event
```

```
array([4321211.03400214,  219950.4291506 ])
```

```
aai_agg
```

```
412832.8602866131
```

Lines

Lines are common geographical representation of transport infrastructure like streets, train tracks or powerlines etc. Here we will play it through for the case of winter storm Lothar's impact on the Dutch Railway System:

Loading Data

Note: Hazard and impact functions data have been loaded above.

```
def gdf_lines():
    gdf_lines = gpd.read_file(Path(DEMO_DIR, "nl_rails.gpkg"))
    gdf_lines = gdf_lines.to_crs(epsg=4326)
    return gdf_lines
```

```
exp_nl_lines = Exposures(gdf_lines())
exp_nl_lines.gdf["impf_WS"] = 1
exp_nl_lines.gdf["value"] = 1
exp_nl_lines.gdf.head()
```

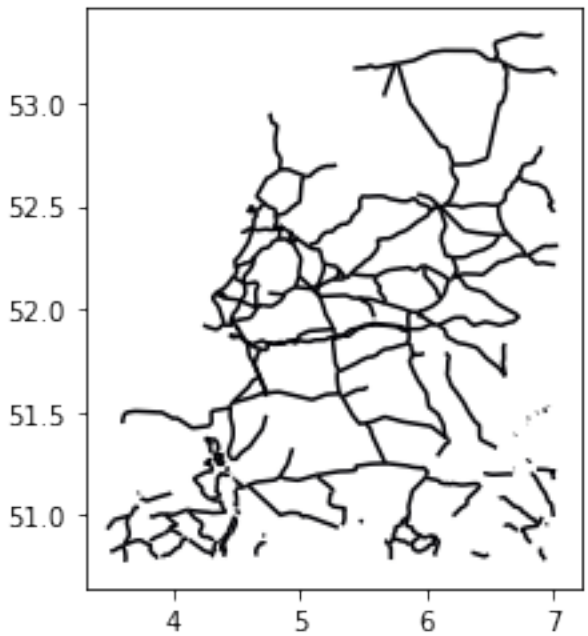
```
       distance                                           geometry  impf_WS  \
0   9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
1  10397.965112  LINESTRING (6.17673 51.35530, 6.17577 51.35410...        1
2  26350.219724  LINESTRING (5.68096 51.25040, 5.68422 51.25020...        1
3  40665.249638  LINESTRING (5.68096 51.25040, 5.67711 51.25030...        1
4   8297.689753  LINESTRING (5.70374 50.85490, 5.70531 50.84990...        1

   value
0      1
```

(continues on next page)

(continued from previous page)

```
1       1
2       1
3       1
4       1
```

```
exp_nl_lines.gdf.plot("value", cmap="inferno");
```



Calculating line impacts - all in one

Example 1: Disaggregate values evenly among road segments; split in points with 0.005 degree distances.

```
imp_deg = u_lp.calc_geom_impact(
    exp=exp_nl_lines,
    impf_set=impf_set,
    haz=storms,
    res=0.005,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
    agg_met=u_lp.AggMethod.SUM,
)
```

```
/Users/ckropf/Documents/Climada/climada_python/climada/util/lines_polys_handler.
↪py:931: UserWarning: Geometry is in a geographic CRS. Results from 'length' are␣
↪likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected␣
↪CRS before this operation.

  line_lengths = gdf_lines.length
```

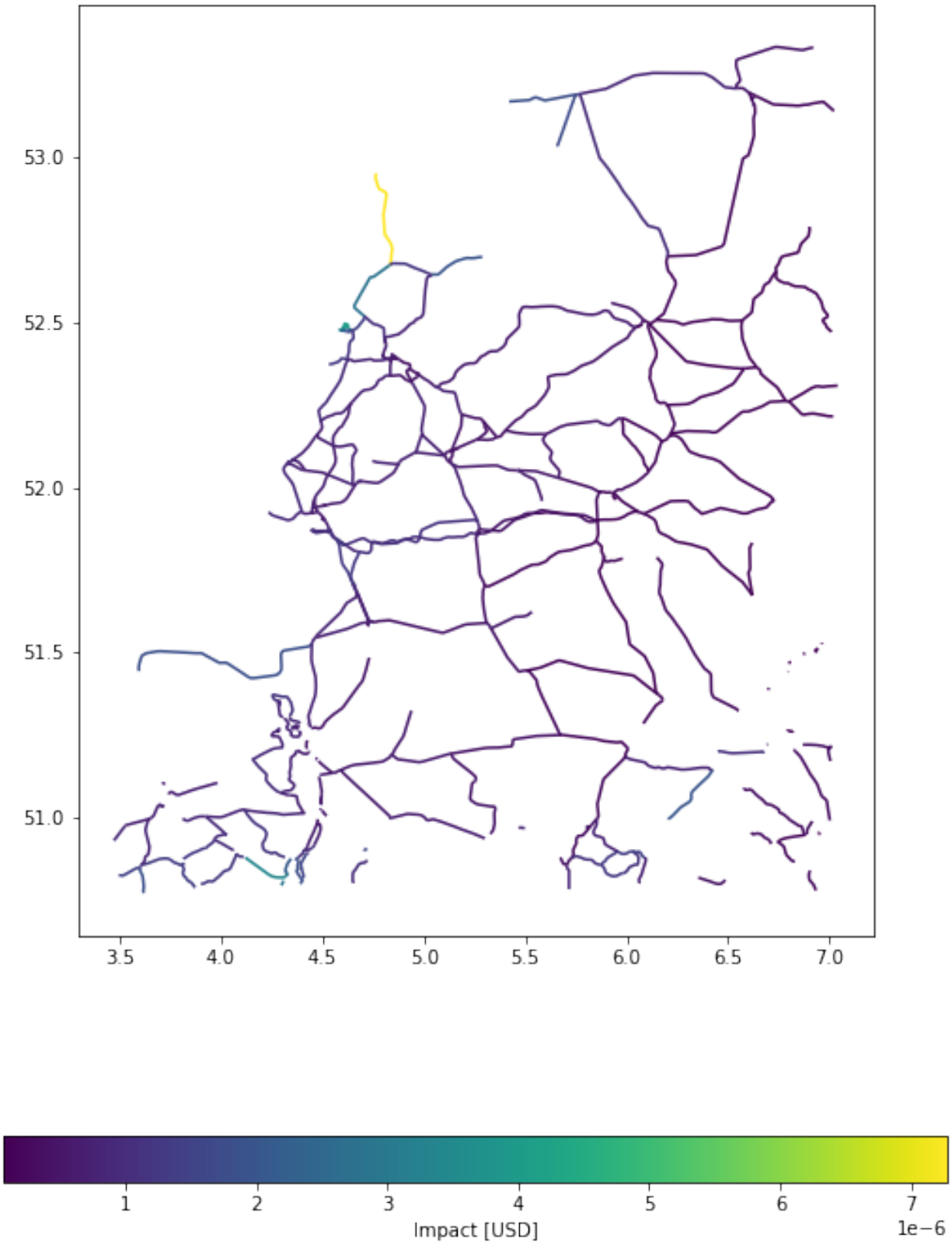
```
2022-06-24 13:16:59,608 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

(continues on next page)

(continued from previous page)

```
2022-06-24 13:16:59,787 - climada.entity.exposures.base - INFO - Matching 10175␣
↪exposures with 9944 centroids.
2022-06-24 13:16:59,789 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:16:59,805 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:16:59,806 - climada.engine.impact - INFO - Calculating damage for 10175␣
↪assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_deg);
```

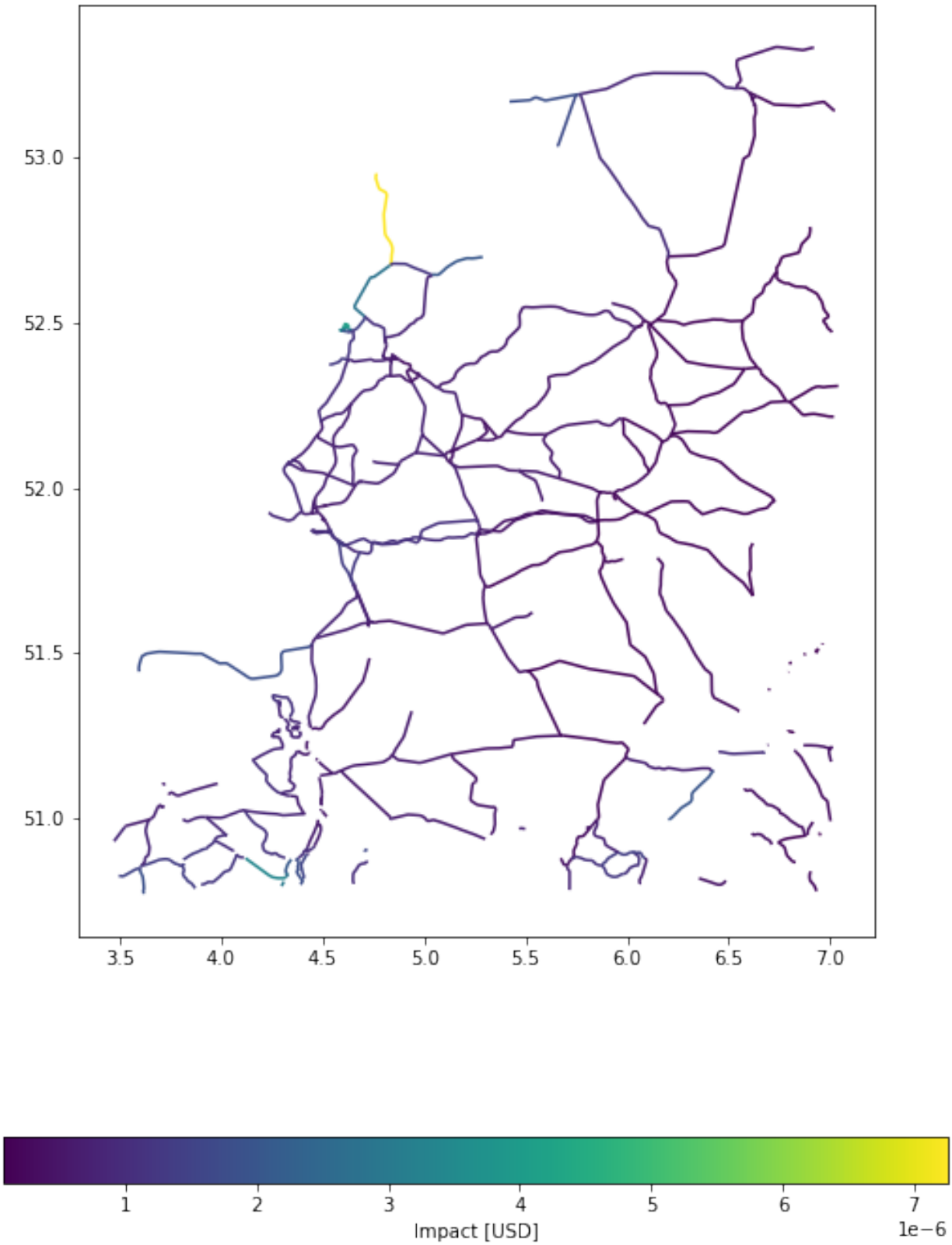


Example 2: Disaggregate values evenly among road segments; split in points with 500 m distances.

```
imp_m = u_lp.calc_geom_impact(
    exp=exp_nl_lines,
    impf_set=impf_set,
    haz=storms,
    res=500,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
    agg_met=u_lp.AggMethod.SUM,
)
```

```
2022-06-24 13:17:00,670 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2022-06-24 13:17:00,827 - climada.entity.exposures.base - INFO - Matching 8399␣
↪exposures with 9944 centroids.
2022-06-24 13:17:00,828 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-06-24 13:17:00,843 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_WS
2022-06-24 13:17:00,845 - climada.engine.impact - INFO - Calculating damage for 8399␣
↪assets (>0) and 2 events.
```

```
u_lp.plot_eai_exp_geom(imp_m);
```




```
import numpy as np
```

(continues on next page)

(continued from previous page)

```
diff = np.max((imp_deg.eai_exp - imp_m.eai_exp) / imp_deg.eai_exp)
print(
    f"The largest relative different between degrees and meters impact in this␣
↪example is {diff}"
)
```

```
The largest relative different between degrees and meters impact in this example is 0.
↪09803913811822067
```

Calculating line impacts - step by step

Step 1: As in the polygon example above, there are several methods to disaggregate line exposures into point exposures, of which several are shown here:

```
# 0.1° distance between points, average value disaggregation
exp_pnt = u_lp.exp_geom_to_pnt(
    exp_nl_lines,
    res=0.1,
    to_meters=False,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
)
exp_pnt.gdf.head()
```

```
2022-06-24 13:17:01,409 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
/Users/ckropf/Documents/Climada/climada_python/climada/util/lines_polys_handler.
↪py:931: UserWarning: Geometry is in a geographic CRS. Results from 'length' are␣
↪likely incorrect. Use 'GeoSeries.to_crs()' to re-project geometries to a projected␣
↪CRS before this operation.

  line_lengths = gdf_lines.length
```

```
         distance                                       geometry_orig  impf_WS  \
0 0   9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  1   9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
1 0  10397.965112  LINESTRING (6.17673 51.35530, 6.17577 51.35410...        1
  1  10397.965112  LINESTRING (6.17673 51.35530, 6.17577 51.35410...        1
  2  10397.965112  LINESTRING (6.17673 51.35530, 6.17577 51.35410...        1

        value                  geometry   latitude  longitude
0 0  0.500000  POINT (6.08850 50.87940)  50.879400   6.088500
  1  0.500000  POINT (6.06079 50.80030)  50.800300   6.060790
1 0  0.333333  POINT (6.17673 51.35530)  51.355300   6.176730
  1  0.333333  POINT (6.12632 51.32440)  51.324399   6.126323
  2  0.333333  POINT (6.08167 51.28460)  51.284600   6.081670
```

```
# 1000m distance between points, no value disaggregation
exp_pnt2 = u_lp.exp_geom_to_pnt(
    exp_nl_lines,
    res=1000,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.FIX,
    disagg_val=None,
)
exp_pnt2.gdf.head()
```

```
2022-06-24 13:17:01,876 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
        distance                                          geometry_orig  impf_WS  \
0 0  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  1  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  2  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  3  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  4  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1

     value                  geometry   latitude  longitude
0 0      1  POINT (6.08850 50.87940)  50.879400   6.088500
  1      1  POINT (6.09416 50.87275)  50.872755   6.094165
  2      1  POINT (6.09161 50.86410)  50.864105   6.091608
  3      1  POINT (6.08744 50.85590)  50.855902   6.087435
  4      1  POINT (6.08326 50.84770)  50.847699   6.083263
```

```
# 1000m distance between points, equal value disaggregation
exp_pnt3 = u_lp.exp_geom_to_pnt(
    exp_nl_lines,
    res=1000,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.DIV,
    disagg_val=None,
)
exp_pnt3.gdf.head()
```

```
2022-06-24 13:17:02,436 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
        distance                                          geometry_orig  impf_WS  \
0 0  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  1  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  2  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  3  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  4  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1

        value                  geometry   latitude  longitude
0 0  0.090909  POINT (6.08850 50.87940)  50.879400   6.088500
  1  0.090909  POINT (6.09416 50.87275)  50.872755   6.094165
  2  0.090909  POINT (6.09161 50.86410)  50.864105   6.091608
```

(continues on next page)

(continued from previous page)

```
  3  0.090909  POINT (6.08744 50.85590)  50.855902   6.087435
  4  0.090909  POINT (6.08326 50.84770)  50.847699   6.083263
```

```
# 1000m distance between points, disaggregation of value according to representative␣
↪distance
exp_pnt4 = u_lp.exp_geom_to_pnt(
    exp_nl_lines,
    res=1000,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.FIX,
    disagg_val=1000,
)
exp_pnt4.gdf.head()
```

```
2022-06-24 13:17:03,116 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```

```
        distance                                        geometry_orig  impf_WS  \
0 0  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  1  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  2  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  3  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1
  4  9414.978692  LINESTRING (6.08850 50.87940, 6.08960 50.87850...        1

     value                  geometry   latitude  longitude
0 0   1000  POINT (6.08850 50.87940)  50.879400   6.088500
  1   1000  POINT (6.09416 50.87275)  50.872755   6.094165
  2   1000  POINT (6.09161 50.86410)  50.864105   6.091608
  3   1000  POINT (6.08744 50.85590)  50.855902   6.087435
  4   1000  POINT (6.08326 50.84770)  50.847699   6.083263
```

Step 2 & 3: The procedure is analogous to the example provided above for polygons.

2.4.4 How to get exposure data from OpenStreetMap

Introduction

OpenStreetMap data is a freely-accessible and valuable data source which can provide information on the geolocations of a variety of assets such as critical infrastructures, buildings, or ecosystems. Such data can then be used within the risk modelling chain of CLIMADA as exposures. In this tutorial we show how to retrieve exposure data by querying the OpenStreetMap database using the osm-flex python package.

Quick example

Here we provide a quick example of an impact calculation with CLIMADA and OpenStreetMap (OSM) data. We use in this example main roads in Honduras as exposures, and historical tropical cyclones as hazard. We load the OSM data using `osm-flex` and disaggregate the exposures, compute the damages, and reaggregate the exposures to their original shape using the function `calc_geom_impact` from the util module `lines_polys_handler`. For more details on the `lines_polys_handler` module, please refer to the documentation.

```
#! Do not copy this cell!
# This cell is only there to remove warnings and render the notebook more readable.
```

(continues on next page)

(continued from previous page)

```
import warnings

warnings.filterwarnings("ignore")
from climada import CONFIG
import logging
from climada.util.config import LOGGER

LOGGER.setLevel(logging.ERROR)
```

```
import matplotlib.pyplot as plt
import osm_flex
import osm_flex.download
import osm_flex.extract

osm_flex.enable_logs()
```

The first step is to download a raw osm.pbf file ("data dump") for Honduras from geofabrik.de and extract the layer of interest (here roads). See the set-up CLIMADA exposures from OpenStreetMap *section* for more details.

```
# (checks if file honduras-latest.osm.pbf already exists)
# file is stored as defined in osm_flex.config.OSM_DATA_DIR unless specified otherwise
iso3 = "HND"
path_ctr_dump = osm_flex.download.get_country_geofabrik(iso3)

# lets extract all roads from the Honduras file, via the wrapper
gdf_roads = osm_flex.extract.extract_cis(path_ctr_dump, "road")

# set crs
gdf_roads = gdf_roads.to_crs(epsg=4326)
```

```
INFO:osm_flex.download:Skip existing file: /Users/user/osm/osm_bpf/honduras-latest.
↪osm.pbf
INFO:osm_flex.extract:query is finished, lets start the loop
extract points: 0it [00:00, ?it/s]
INFO:osm_flex.extract:query is finished, lets start the loop
extract multipolygons: 0it [00:06, ?it/s]
INFO:osm_flex.extract:query is finished, lets start the loop
extract lines: 100%|██████████| 132099/132099 [00:06<00:00, 21926.39it/s]
```

Next, we set up the exposure, and select our hazard and vulnerability.

```
from climada.util.api_client import Client
import climada.util.lines_polys_handler as u_lp
from climada.entity.impact_funcs import ImpactFuncSet
from climada.entity import ImpfTropCyclone
from climada.entity import Exposures

# load observed tropical cyclones for Honduras from data API
haz = Client().get_hazard(
    "tropical_cyclone", properties={"country_iso3alpha": iso3, "event_type": "observed
↪"}
)
```

(continues on next page)

(continued from previous page)

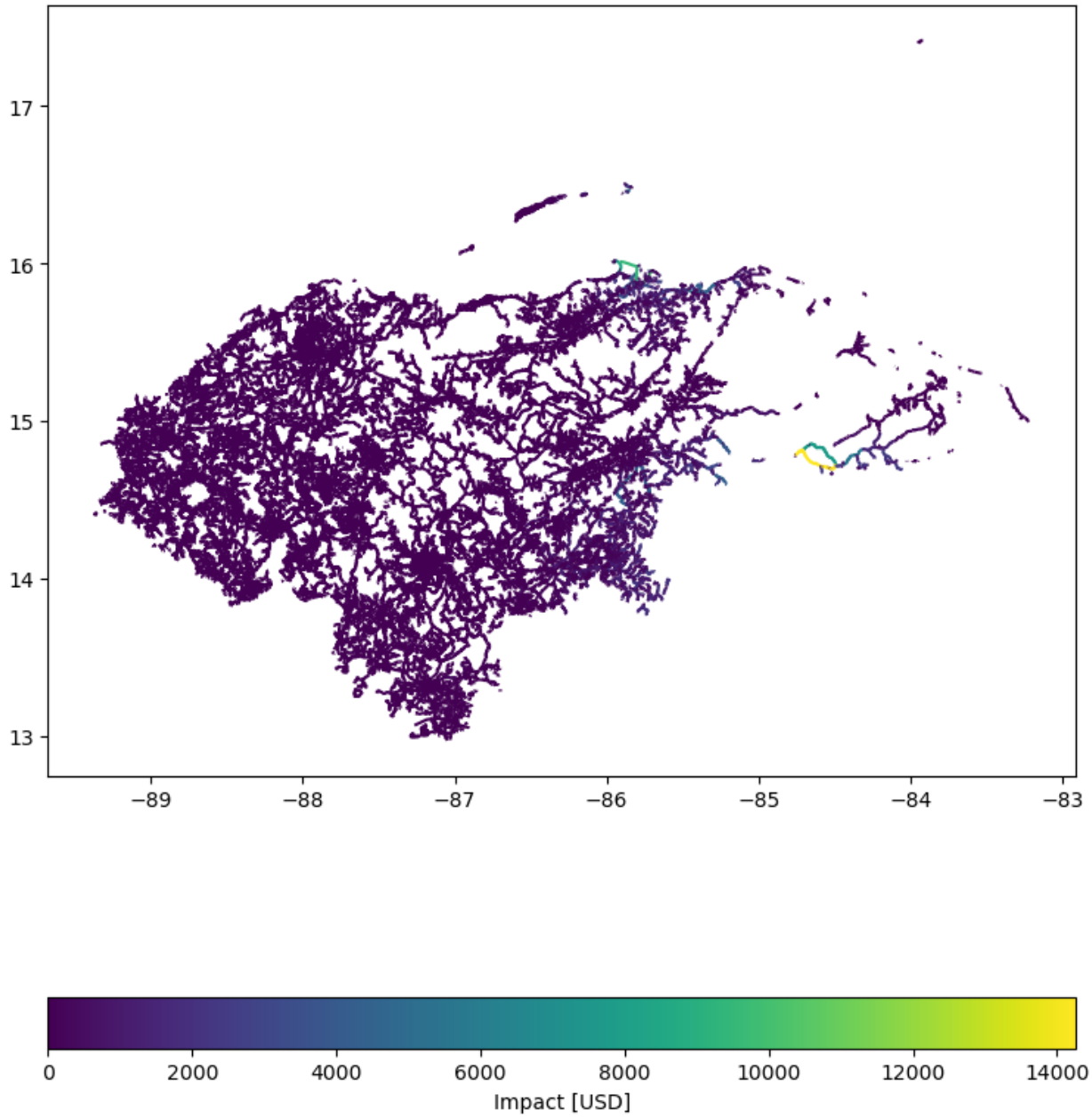
```
# exposures
exp_line = Exposures(gdf_roads)

# impact function
impf_line = ImpfTropCyclone.from_emanuel_usa()
impf_set = ImpactFuncSet([impf_line])

exp_line.data["impf_TC"] = 1  # specify impact function
```

Finally, we use the wrapper function `calc_geom_impact` to compute the impacts in one line of code. For a reminder, the `calc_geom_impact` is covering the 3 steps of shapes-to-points disagreggation, impact calculation, and reaggregation to the original shapes. `calc_geom_impact` requires the user to specify a target resolution of the disaggregation (`res`), as well as how to assign a value to the disaggregated exposure (`disagg_met` and `disagg_val`). Here, we arbitrarily decide to give a fixed value 100k USD to each road segment of 500m, but note that other options are possible.

```
# disaggregate in the same CRS as the exposures are defined (here meters), resolution␣
↪500m
# replicate values on points
# aggregate by summing

impact = u_lp.calc_geom_impact(
    exp=exp_line,
    impf_set=impf_set,
    haz=haz,
    res=500,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.FIX,
    disagg_val=1e5,
    agg_met=u_lp.AggMethod.SUM,
);
```

Finally, let's plot the impact calculated.

```
# plot the calculated impacts
u_lp.plot_eai_exp_geom(impact);
```



Set-up CLIMADA exposures from OpenStreetMap

Within the CLIMADA plateform, there are two main ways to obtain exposure data from OpenStreetMap:

1. Using the `osm-flex` module directly available from the CLIMADA core environment
2. Using the `OSMApiQuery` methods from the `Exposures.osm_dataloader` module available in CLIMADA petals

In this tutorial, we will only provide a brief introduction to the first method making use of `osm-flex`. Please refer to the documentation for more detailed explanations on the two methods.

osm-flex

`osm-flex` is a python package which allows to flexibly extract data from OpenStreetMap. See osm-flex and the associated publication for more information: *Mühlhofer, Kropf, Riedel, Bresch and Koks: OpenStreetMap for Multi-Faceted Climate Risk Assessments. Environ. Res. Commun.* ***6*** *015005 doi: 10.1088/2515-7620/ad15ab* Obtaining CLIMADA exposures object from OpenStreetMap using `osm-flex` consists in the following steps:

1. **Download** a raw osm.pbf file ("data dump") for a specific country or region from geofabrik.de
2. **Extract** the features of interest (e.g. a road network) as a geodataframe
3. **Pre-process**; apply pre-processing steps as e.g. clipping, simplifying, or reprojecting the retrieved layer.
4. **Cast** the geodataframe into a CLIMADA `Exposures` object.
5. **Disagreggate** complex shapes exposures into points for impact calculation.

Once those 5 steps are completed, one can proceed with the impact calculation. For more details on how to use lines and polygons as exposures within CLIMADA, please refer to the documentation.

In the following, we illustrate how to obtain different exposures types such as forests or healthcare facilities, and how to use them within CLIMADA as points, lines, and polygons exposures. We also briefly illustrate the use of the `simplify` module available within the `osm-flex` package.

Download a raw osm.pbf file ("data dump")

First, we need to select a specific country and download its data from geofabrik.de. It is possible to download data from specific countries using iso3 codes or from regions directly.

```
# (checks if file honduras-latest.osm.pbf already exists)
# file is stored as defined in osm_flex.config.OSM_DATA_DIR unless specified otherwise
iso3 = "HND"
path_ctr_dump = osm_flex.download.get_country_geofabrik(iso3)
```

```
INFO:osm_flex.download:Skip existing file: /Users/user/osm/osm_bpf/honduras-latest.
↪osm.pbf
```

Extract the features of interest

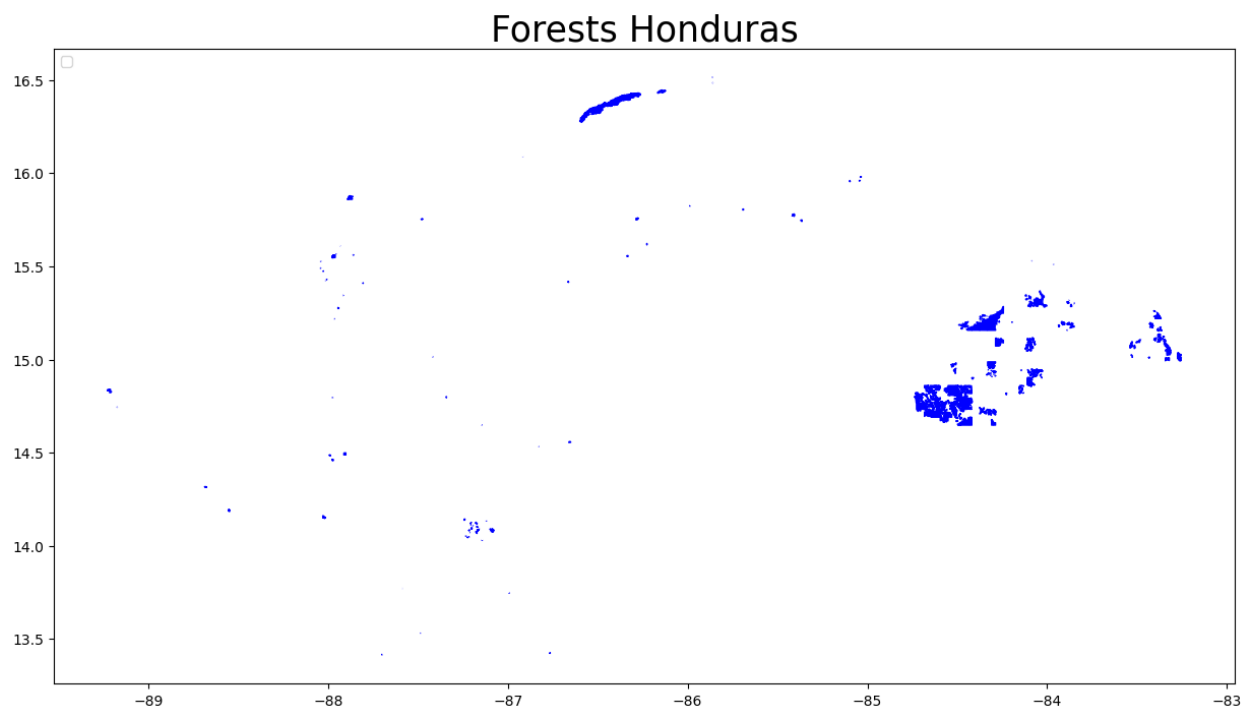
We next extract the exposures data of interest from OSM using the `extract()` method which allows us to query any tags available on OpenStreetMap. Two variables have to be specified: `osm_keys`, a list with all the columns to report in the GeoDataFrame, and `osm_query`, a string of key-value constraints to apply during the search. We illustrate its use by querying the download of forests for Honduras.

```
# Let us first extract forests (multipolygons) in Honduras
osm_keys = ["landuse"]
osm_query = "landuse='forest'"
gdf_forest = osm_flex.extract.extract(
    path_ctr_dump, "multipolygons", osm_keys, osm_query
)
```

```
INFO:osm_flex.extract:query is finished, lets start the loop
extract multipolygons: 100%|██████████| 750/750 [00:06<00:00, 124.46it/s]
```

```
# set crs
gdf_forest = gdf_forest.to_crs(epsg=4326)
```

```
# Plot results
ax = gdf_forest.plot(
    figsize=(15, 15),
    alpha=1,
    markersize=5,
    color="blue",
    edgecolor="blue",
    label="forests HND",
)
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles=handles, loc="upper left")
ax.set_title("Forests Honduras", fontsize=25)
plt.show()
```



Alternatively, we can use the `extract_cis` method to download specific types of critical infrastructures available on OSM.

```
# check available critical infrastructure types:
osm_flex.config.DICT_CIS_OSM.keys()
```

```
dict_keys(['education', 'healthcare', 'water', 'telecom', 'road', 'main_road', 'rail',
↪ 'air', 'gas', 'oil', 'power', 'wastewater', 'food', 'buildings'])
```

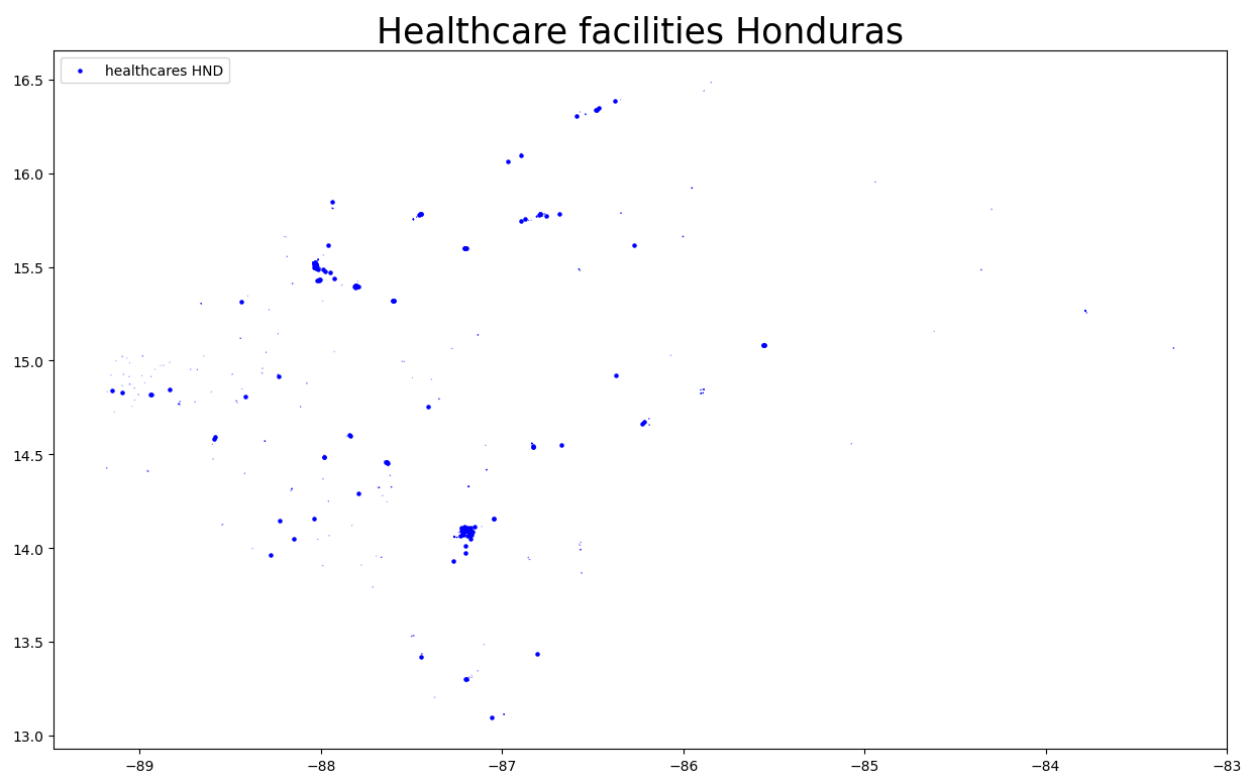
```
# lets extract all healthcares from the Honduras file, via the wrapper
gdf_hc = osm_flex.extract.extract_cis(path_ctr_dump, "healthcare")
```

```
INFO:osm_flex.extract:query is finished, lets start the loop
```

```
extract points: 100%|██████████| 149/149 [00:00<00:00, 344.56it/s]
INFO:osm_flex.extract:query is finished, lets start the loop
extract multipolygons: 100%|██████████| 330/330 [00:11<00:00, 29.06it/s]
```

```
# set crs
gdf_hc = gdf_hc.to_crs(epsg=4326)
```

```
# plot results
ax = gdf_hc.plot(
    figsize=(15, 15),
    alpha=1,
    markersize=5,
    color="blue",
    edgecolor="blue",
    label="healthcares HND",
)
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles=handles, loc="upper left")
ax.set_title("Healthcare facilities Honduras", fontsize=25)
plt.show()
```



Pre-process the retrieved OSM exposures data (optional)

It can be necessary to apply some preprocessing steps before using the retrieved OSM data as CLIMADA exposures. In particular, the following two pre-processing tasks are available as modules within the `osm-flex` package:

1. **Clipping**; allows to clip the country data to a user-determined region.

2. **Simplify**; in some cases, simplifying the retrieved data is necessary to remove redundant or erroneous

In the following, we show how to simplify some data retrieved from OpenStreetMap using the `osm_flex.simplify` module. For more details on clipping or on other features available within `osm_flex`, please refer to its documentation.

```
# here we illustrate how to simplify the polygon-based forest layer by removing small␣
↪polygons
import osm_flex.simplify as sy

# initial number of points
print(f"Number of results: {len(gdf_forest)}")

gdf_forest = gdf_forest.to_crs("epsg:5456")  # metre-based CRS for Honduras
min_area = 100

gdf_forest = sy.remove_small_polygons(
    gdf_forest, min_area
)  # remove all areas < 100m2 (always in units of respective CRS)
print(f"Number of results after removal of small polygons: {len(gdf_forest)}")
```

```
Number of results: 750
Number of results after removal of small polygons: 739
```

Cast OSM exposures into CLIMADA exposures objects

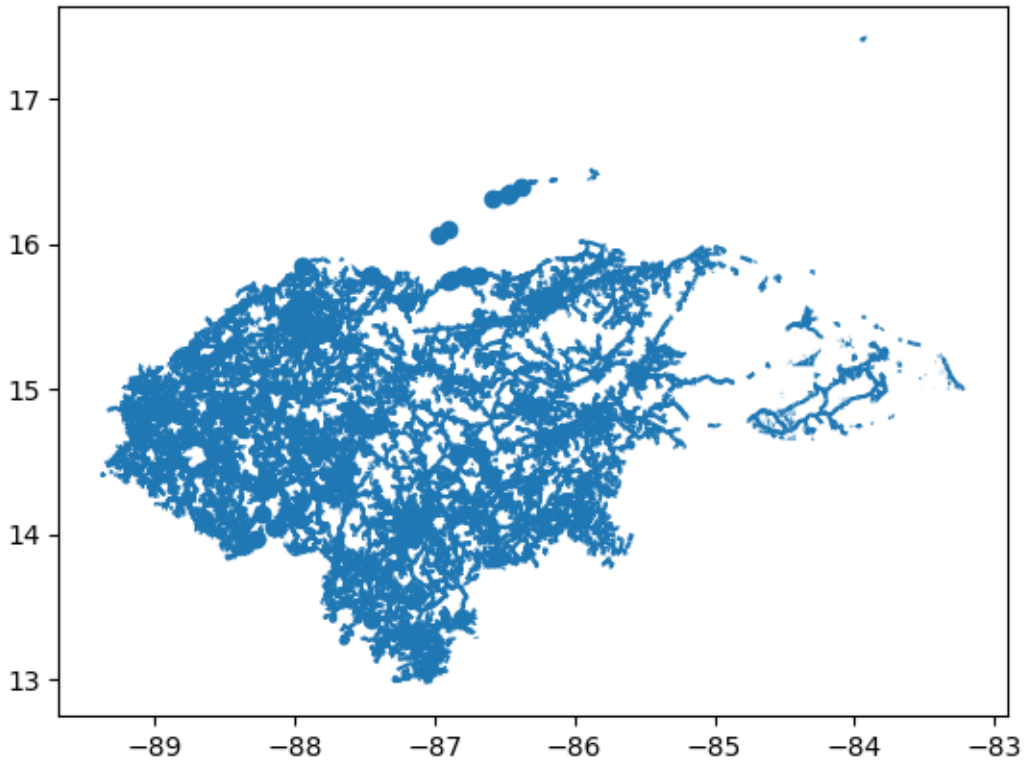
The last step consists in transforming exposures data obtained from OSM into CLIMADA-readable objects. This is simply done using the CLIMADA `Exposures`class.

```
from climada.entity import Exposures

gdf_forest = gdf_forest.to_crs(
    epsg=4326
)  # !ensure that all exposures are in the same CRS!
exp_poly = Exposures(gdf_forest)
```

Additionally, multiple exposures of different types can be combine within a single CLIMADA `Exposures` object using `concat`.

```
exp_points = Exposures(gdf_hc)

exp_mix = Exposures.concat([exp_points, exp_line, exp_poly])
```

```
exp_mix.plot()
```



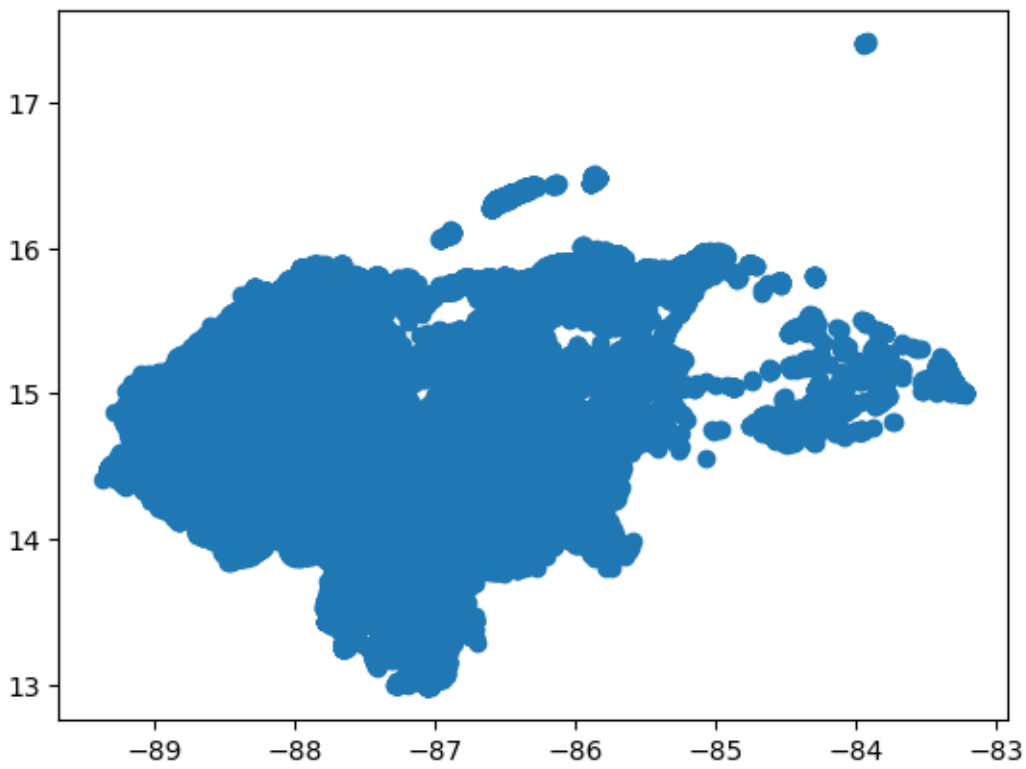
Disagreggate complex shapes into point data

The last step before proceeding to the usual impact calculation consists in transforming all the exposure data that is in a format other than point (e.g. lines, polygons) into point data (disagreggation) and assigning them values. Those two tasks can be done simultaneously using the util function `exp_geom_to_pnt`. Disagreggating and assigning values to the disagreggated exposures requires the following:

1. Specify a resolution for the disaggregation (`res`).
2. Specify a value to be disaggregated (`disagg_val`).
3. Specify how to distribute the value to the disaggregated points (`disagg_met`).

In the following, we illustrate how to disaggregate our mixed-types exposures to a 10km-resolution, arbitrarily assigning a fixed value of 500k USD to each point. For more details on how to use lines and polygons as exposures within CLIMADA, please refer to the documentation.

```
exp_mix_pnt = u_lp.exp_geom_to_pnt(
    exp_mix,
    res=10000,
    to_meters=True,
    disagg_met=u_lp.DisaggMethod.FIX,
    disagg_val=5e5,
)
exp_mix_pnt.plot()
```



2.5 Impact Tutorials

These tutorials show how to compute impacts with CLIMADA, and all related aspects such as impact functions, adaptation measures and discount rates and cost-benefit analysis.

The first tutorial presents an end-to-end impact calculation, and subsequent ones present each aspect in more details.

Additionally you can find a guide on how to populate impact data from EM-DAT database.

2.5.1 END-TO-END IMPACT CALCULATION

Goal of this tutorial

The goal of this tutorial is to show a full end-to-end impact computation. Note that this tutorial exemplifies the work flow, but does not explore all possible features.

What is an Impact?

The impact is the combined effect of hazard events on a set of exposures mediated by a set of impact functions. By computing the impact for each event (historical and synthetic) and for each exposure value at each geographical location, the Impact class provides different risk measures, such as the expected annual impact per exposure, the probable maximum impact for different return periods, and the total average annual impact.

Impact class data structure

The impact class does not require any attributes to be defined by the user. All attributes are set by the method `ImpactCalc.impact()`. This method requires three attributes: an `Exposure`, a `Hazard`, and an `ImpactFuncSet`. After calling `ImpactCalc(Exposure, ImpactFuncSet, Hazard).impact(save_mat=False)`, the Impact object has the following attributes:

Attributes from input	Data Type	Description
event_id	list(int)	id (>0) of each hazard event (Hazard.event_id)
event_name	(list(str))	name of each event (Hazard.event_name)
date	np.array	date of events (Hazard.date)
coord_exp	np.array	exposures coordinates [lat, lon] (in degrees) (Exposure.latidue, Exposure.longitude)
frequency	np.array	frequency of events (Hazard.frequency)
frequency_unit	str	unit of event frequency, by default '1/year', i.e., annual (Hazard.frequency_unit)
unit	str	value unit used (Exposure.value_unit)
csr	str	unit system for Exposure and Hazard geographical data (Exposure.csr)

Computed attributes	Data Type	Description
at_event	np.array	impact for each hazard event summed over all locations
eai_exp	np.array	expected annual impact for each locations, summed over all events weigthed by frequency
aai_agg	float	total annual average aggregated impact value (summed over events and locations)
impt_mat	sparse.csr_	This is the impact matrix. This matrix has the events as rows and the exposure points as columns (num_events X num_exp). It will only be filled with impact values if save_mat is True.
tot_value	float	total exposure value affected (sum of value all exposures locations affected by at least one hazard event)

All other methods compute values from the attributes set by `ImpactCalc.impact()`. For example, one can compute the frequency exceedance curve, plot impact data, or compute traditional risk transfer over impact.

How do I compute an impact in CLIMADA?

In CLIMADA, impacts are computed using the `Impact` class. The computation of the impact requires an `Exposure` , an `ImpactFuncSet`, and a `Hazard` object. For details about how to define ***Exposures*** , ***Hazard***, ***Impact Functions*** see the respective tutorials.

The steps of an impact caculations are typically:

- Set exposure
- Set hazard and hazard centroids
- Set impact functions in impact function set
- Compute impact
- Visualize, save, use impact output

Hints: Before computing the impact of a given `Exposure` and `Hazard`, it is important to correctly match the `Exposures` coordinates with the `Hazard Centroids`. Try to have similar resolutions in `Exposures` and `Hazard`. By the impact calculation the nearest neighbor for each `Exposure` to the `Hazard's Centroids` is searched.

Hint: Set first the `Exposures` and use its coordinates information to set a matching `Hazard`.

Hint: The configuration value `max_matrix_size` controls the maximum matrix size contained in a chunk. By default it is set to 1e9 in the default config file. A high value makes the computation fast at the cost of increased memory consumption. You can decrease its value if you are having memory issues with the `ImpactCalc.impact()` method. (See the *config guide* on how to set configuration values).

Detailed Impact calculation - LitPop + TropCyclone

We present a detailed example for the hazard ***Tropical Cyclones*** and the exposures from ***LitPop*** .

Define the exposure

Reminder: The exposures must be defined according to your problem either using CLIMADA exposures such as **BlackMarble**, ***LitPop***, **OSM**, extracted from external sources (imported via csv, excel, api, …) or directly user defined.

As a reminder, exposures are geopandas dataframes with at least columns 'latitude', 'longitude' and 'value' of exposures.

For impact calculations, for each exposure values of the corresponding impact function to use (defined by the column `impf_`) and the associated hazard centroids must be defined. This is done after defining the impact function(s) and the hazard(s).

See tutorials on ***Exposures*** , ***Hazard***, ***ImpactFuncSet*** for more details.

Exposures are either defined as a series of (latitude/longitude) points or as a raster of (latitude/longitude) points. Fundamentally, this changes nothing for the impact computations. Note that for larger number of points, consider using a raster which might be more efficient (computationally). For a low number of points, avoid using a raster if this adds a lot of exposures values equal to 0.

We shall here use a raster example.

```
# Exposure from the module Litpop
# Note that the file gpw_v4_population_count_rev11_2015_30_sec.tif must be downloaded␣
↪(do not forget to unzip) if
# you want to execute this cell on your computer. If you haven't downloaded it before,
↪ please have a look at the section
# "population count" of the LitPop tutorial.

%matplotlib inline
import numpy as np
from climada.entity import LitPop

# Cuba with resolution 10km and financial_mode = income group.
exp_lp = LitPop.from_countries(
    countries=["CUB"], res_arcsec=300, fin_mode="income_group"
)
exp_lp.check()
```

```
2023-01-26 11:57:14,980 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2023-01-26 11:57:15,071 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,073 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,130 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,132 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,193 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

(continues on next page)

(continued from previous page)

```
2023-01-26 11:57:15,195 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,305 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,307 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,371 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,373 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,410 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,412 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,413 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,444 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,445 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,448 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,488 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,488 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,491 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,521 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:15,523 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,525 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,603 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,605 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,661 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,663 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,724 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,725 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,791 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,792 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,826 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
```

(continues on next page)

(continued from previous page)

```
2023-01-26 11:57:15,828 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,831 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,892 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,894 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:15,958 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:15,959 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,033 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,035 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,771 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,773 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,831 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,833 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,866 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:16,868 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,871 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:16,939 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:16,941 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,005 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,007 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,066 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,067 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,127 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,130 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,163 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,166 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,169 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2023-01-26 11:57:17,197 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,199 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,202 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,269 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,271 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,334 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,336 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

```
2023-01-26 11:57:17,411 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,413 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,445 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,448 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,451 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,476 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,477 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,479 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,546 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,550 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,582 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:17,584 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,585 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,645 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,646 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,702 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,704 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,769 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,771 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2023-01-26 11:57:17,837 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,840 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,924 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:17,925 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:17,998 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:18,000 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:18,066 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:57:18,068 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:57:18,100 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:57:18,576 - climada.util.finance - INFO - GDP CUB 2018: 1.000e+11.
2023-01-26 11:57:18,671 - climada.util.finance - INFO - Income group CUB 2018: 3.
2023-01-26 11:57:18,692 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:57:18,694 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:57:18,695 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:57:18,697 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:57:18,699 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:57:18,700 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:57:18,702 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:57:18,703 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:57:18,704 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:57:18,706 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
exp_lp.gdf.head()
```

```
          value                     geometry   latitude  longitude  region_id  \
0  1.077368e+05  POINT (-81.37500 21.70833)  21.708333 -81.375000        192
1  1.671873e+06  POINT (-81.54167 21.62500)  21.625000 -81.541667        192
2  3.421208e+06  POINT (-82.95833 21.87500)  21.875000 -82.958333        192
3  1.546590e+07  POINT (-82.87500 21.87500)  21.875000 -82.875000        192
4  7.168305e+07  POINT (-82.79167 21.87500)  21.875000 -82.791667        192

   impf_
0      1
1      1
2      1
3      1
4      1
```

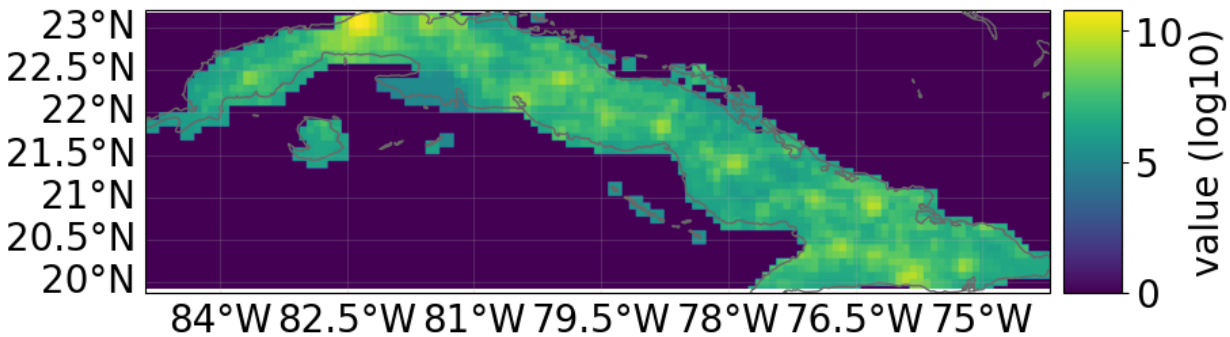
```
# not needed for impact calculations
```

(continues on next page)

(continued from previous page)

```
# visualize the define exposure
exp_lp.plot_raster()
print("\n Raster properties exposures:", exp_lp.meta)
```

```
2023-01-26 11:57:18,777 - climada.util.coordinates - INFO - Raster from resolution 0.
↪08333332999999854 to 0.08333332999999854.

 Raster properties exposures: {'width': 129, 'height': 41, 'crs': <Geographic 2D CRS:␣
↪EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
, 'transform': Affine(0.08333333000000209, 0.0, -84.91666666500001,
       0.0, -0.08333332999999854, 23.249999994999996)}
```



Define the hazard

Let us define a tropical cyclone hazard using the TropCyclone and TCTracks modules.

```
from climada.hazard import TCTracks, TropCyclone, Centroids

# Load histrocial tropical cyclone tracks from ibtracs over the North Atlantic basin␣
↪between 2010-2012
ibtracks_na = TCTracks.from_ibtracs_netcdf(
    provider="usa", basin="NA", year_range=(2010, 2012), correct_pres=True
)
print("num tracks hist:", ibtracks_na.size)

ibtracks_na.equal_timestep(
    0.5
)  # Interpolation to make the track smooth and to allow applying calc_perturbed_
↪trajectories
# Add randomly generated tracks using the calc_perturbed_trajectories method (1 per␣
↪historical track)
```

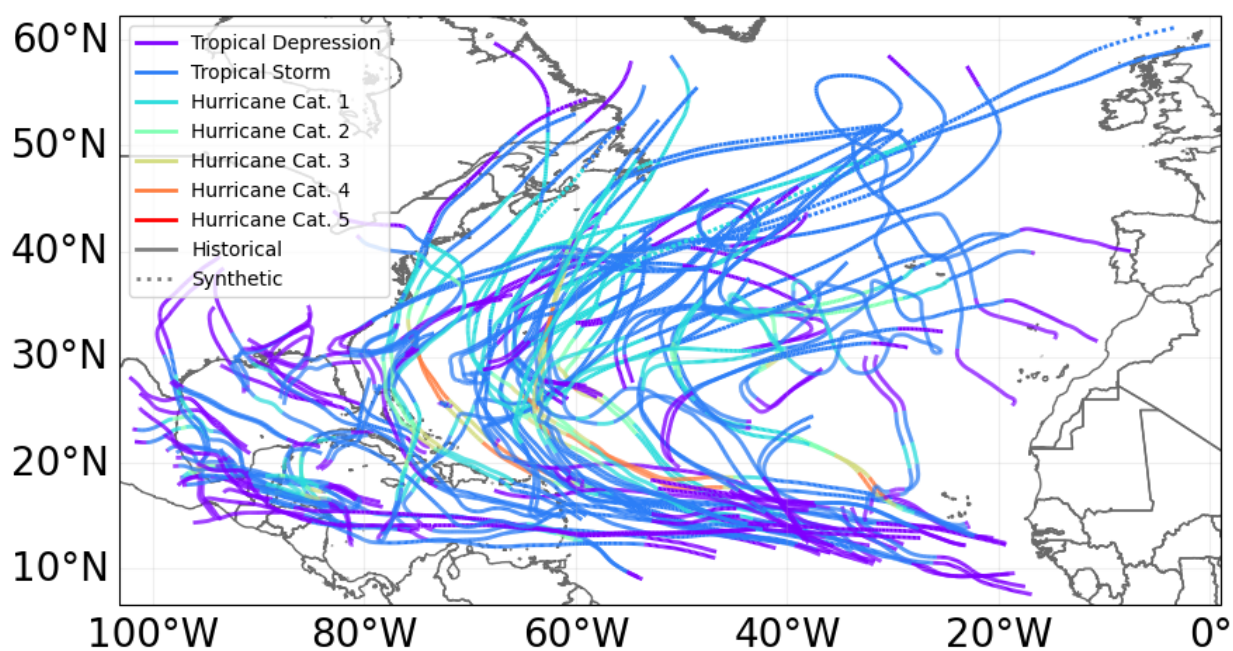
(continues on next page)

(continued from previous page)

```
ibtracks_na.calc_perturbed_trajectories(nb_synth_tracks=1)
print("num tracks hist+syn:", ibtracks_na.size)
```

```
2023-01-26 11:57:25,332 - climada.hazard.tc_tracks - WARNING - `correct_pres` is␣
↪deprecated. Use `estimate_missing` instead.
2023-01-26 11:57:26,701 - climada.hazard.tc_tracks - INFO - Progress: 10%
2023-01-26 11:57:26,905 - climada.hazard.tc_tracks - INFO - Progress: 20%
2023-01-26 11:57:27,075 - climada.hazard.tc_tracks - INFO - Progress: 30%
2023-01-26 11:57:27,229 - climada.hazard.tc_tracks - INFO - Progress: 40%
2023-01-26 11:57:27,390 - climada.hazard.tc_tracks - INFO - Progress: 50%
2023-01-26 11:57:27,542 - climada.hazard.tc_tracks - INFO - Progress: 60%
2023-01-26 11:57:27,713 - climada.hazard.tc_tracks - INFO - Progress: 70%
2023-01-26 11:57:27,876 - climada.hazard.tc_tracks - INFO - Progress: 80%
2023-01-26 11:57:28,032 - climada.hazard.tc_tracks - INFO - Progress: 90%
2023-01-26 11:57:28,197 - climada.hazard.tc_tracks - INFO - Progress: 100%
num tracks hist: 60
2023-01-26 11:57:28,229 - climada.hazard.tc_tracks - INFO - Interpolating 60 tracks␣
↪to 0.5h time steps.
2023-01-26 11:57:32,223 - climada.hazard.tc_tracks_synth - INFO - Computing 60␣
↪synthetic tracks.
num tracks hist+syn: 120
```

```
# not needed for calculations
# visualize tracks
ax = ibtracks_na.plot()
ax.get_legend()._loc = 2
```



From the tracks, we generate the hazards (the tracks are only the coordinates of the center of the cyclones, the full cyclones however affects a region around the tracks).

First thing we define the set of centroids which are geographical points where the hazard has a defined value. In our case,

we want to define windspeeds from the tracks.

Remember: In the impact computations, for each exposure geographical point, one must assign a centroid from the hazard. By default, each exposure is assigned to the closest centroid from the hazard. But one can also define manually which centroid is assigned to which exposure point.

Examples:

- Define the exposures from a given source (e.g., raster of asset values from LitPop). Define the hazard centroids from the exposures' geolocations (e.g. compute Tropical Cyclone windspeed at each raster point and assign centroid to each raster point).
- Define the exposures from a given source (e.g. houses position and value). Define the hazard from a given source (e.g. where lanslides occur). Use a metric to assign to each exposures point a hazard centroid (all houses in a radius of 5km around the lanslide are assigned to this centroid, if a house is within 5km of two landslides, choose the closest one).
- Define a geographical raster. Define the exposures value on this raster. Define the hazard centroids on the geographical raster.

We shall pursue with the first case (Litpop + TropicalCyclone)

Hint: computing the wind speeds in many locations for many tc tracks is a computationally costly operation. Thus, we should define centroids only where we also have an exposure.

```
# Define the centroids from the exposures position
lat = exp_lp.gdf["latitude"].values
lon = exp_lp.gdf["longitude"].values
centrs = Centroids.from_lat_lon(lat, lon)
centrs.check()
```

```
# Using the tracks, compute the windspeed at the location of the centroids
tc = TropCyclone.from_tracks(ibtracks_na, centroids=centrs)
tc.check()
```

```
2023-01-26 11:58:43,439 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2023-01-26 11:58:44,363 - climada.util.coordinates - INFO - dist_to_coast: UTM 32616␣
↪(1/3)
2023-01-26 11:58:44,606 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617␣
↪(2/3)
2023-01-26 11:58:45,335 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618␣
↪(3/3)
2023-01-26 11:58:45,767 - climada.hazard.trop_cyclone - INFO - Mapping 120 tracks to␣
↪1388 coastal centroids.
2023-01-26 11:58:46,030 - climada.hazard.trop_cyclone - INFO - Progress: 10%
2023-01-26 11:58:46,203 - climada.hazard.trop_cyclone - INFO - Progress: 20%
2023-01-26 11:58:46,499 - climada.hazard.trop_cyclone - INFO - Progress: 30%
2023-01-26 11:58:46,784 - climada.hazard.trop_cyclone - INFO - Progress: 40%
2023-01-26 11:58:47,231 - climada.hazard.trop_cyclone - INFO - Progress: 50%
2023-01-26 11:58:47,353 - climada.hazard.trop_cyclone - INFO - Progress: 60%
2023-01-26 11:58:47,546 - climada.hazard.trop_cyclone - INFO - Progress: 70%
2023-01-26 11:58:47,667 - climada.hazard.trop_cyclone - INFO - Progress: 80%
2023-01-26 11:58:48,102 - climada.hazard.trop_cyclone - INFO - Progress: 90%
2023-01-26 11:58:48,418 - climada.hazard.trop_cyclone - INFO - Progress: 100%
```

Hint: The operation of computing the windspeed in different location is in general computationally expensive. Hence,

if you have a lot of tropical cyclone tracks, you should first make sure that all your tropical cyclones actually affect your exposure (remove those that don't). Then, be careful when defining the centroids. For a large country like China, there is no need for centroids 500km inland (no tropical cyclones gets so far).

Impact function

For Tropical Cyclones, some calibrated default impact functions exist. Here we will use the one from Emanuel (2011).

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone

# impact function TC
impf_tc = ImpfTropCyclone.from_emanuel_usa()

# add the impact function to an Impact function set
impf_set = ImpactFuncSet([impf_tc])
impf_set.check()
```

Recall that the exposures, hazards and impact functions must be matched in the impact calculations. Here it is simple, since there is a single impact function for all the hazards. We must simply make sure that the exposure is assigned this impact function through renaming the `impf\_` column from the hazard type of the impact function in the impact function set and set the values of the column to the id of the impact function.

```
# Get the hazard type and hazard id
[haz_type] = impf_set.get_hazard_types()
[haz_id] = impf_set.get_ids()[haz_type]
print(f"hazard type: {haz_type}, hazard id: {haz_id}")
```

```
hazard type: TC, hazard id: 1
```

```
# Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + haz_type}, inplace=True)
exp_lp.gdf["impf_" + haz_type] = haz_id
exp_lp.check()
exp_lp.gdf.head()
```

```
2023-01-26 11:58:48,792 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:58:48,792 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:58:48,792 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:58:48,792 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
          value                    geometry   latitude  longitude  region_id  \
0  1.077368e+05  POINT (-81.37500 21.70833)  21.708333 -81.375000        192
1  1.671873e+06  POINT (-81.54167 21.62500)  21.625000 -81.541667        192
2  3.421208e+06  POINT (-82.95833 21.87500)  21.875000 -82.958333        192
3  1.546590e+07  POINT (-82.87500 21.87500)  21.875000 -82.875000        192
4  7.168305e+07  POINT (-82.79167 21.87500)  21.875000 -82.791667        192

   impf_TC
0        1
1        1
2        1
3        1
4        1
```

Impact computation

We are finally ready for the impact computation. This is the simplest step. Just give the exposure, impact function and hazard to the ImpactCalc.impact() method.

Note: we did not specifically assign centroids to the exposures. Hence, the default is used - each exposure is associated with the closest centroids. Since we defined the centroids from the exposures, this is a one-to-one mapping.

Note: we did not define an `Entity` in this impact calculations. Recall that `Entity` is a container class for ***Exposures***, ***Impact Functions***, ***Discount Rates*** and ***Measures***. Since we had only one Exposure and one Impact Function, the container would not have added any value, but for more complex projects, the Entity class is very useful.

```
# Compute impact
from climada.engine import ImpactCalc

imp = ImpactCalc(exp_lp, impf_set, tc).impact(
    save_mat=False
)  # Do not save the results geographically resolved (only aggregate values)
```

```
2023-01-26 11:58:48,877 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 1388 centroids.
2023-01-26 11:58:48,877 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪4164 assets (>0) and 120 events.
```

```
exp_lp.gdf
```

```
             value                    geometry   latitude  longitude  \
0     1.077368e+05  POINT (-81.37500 21.70833)  21.708333 -81.375000
1     1.671873e+06  POINT (-81.54167 21.62500)  21.625000 -81.541667
2     3.421208e+06  POINT (-82.95833 21.87500)  21.875000 -82.958333
3     1.546590e+07  POINT (-82.87500 21.87500)  21.875000 -82.875000
4     7.168305e+07  POINT (-82.79167 21.87500)  21.875000 -82.791667
...            ...                         ...         ...        ...
1383  1.496797e+06  POINT (-78.62500 22.54167)  22.541667 -78.625000
1384  5.387835e+07  POINT (-78.37500 22.54167)  22.541667 -78.375000
1385  4.077093e+06  POINT (-78.45833 22.45833)  22.458333 -78.458333
1386  2.249377e+07  POINT (-78.37500 22.45833)  22.458333 -78.375000
1387  6.191982e+06  POINT (-78.29167 22.45833)  22.458333 -78.291667

      region_id  impf_TC  centr_TC
0           192        1         0
1           192        1         1
2           192        1         2
3           192        1         3
4           192        1         4
...         ...      ...       ...
1383        192        1      1383
1384        192        1      1384
1385        192        1      1385
1386        192        1      1386
1387        192        1      1387

[1388 rows x 7 columns]
```

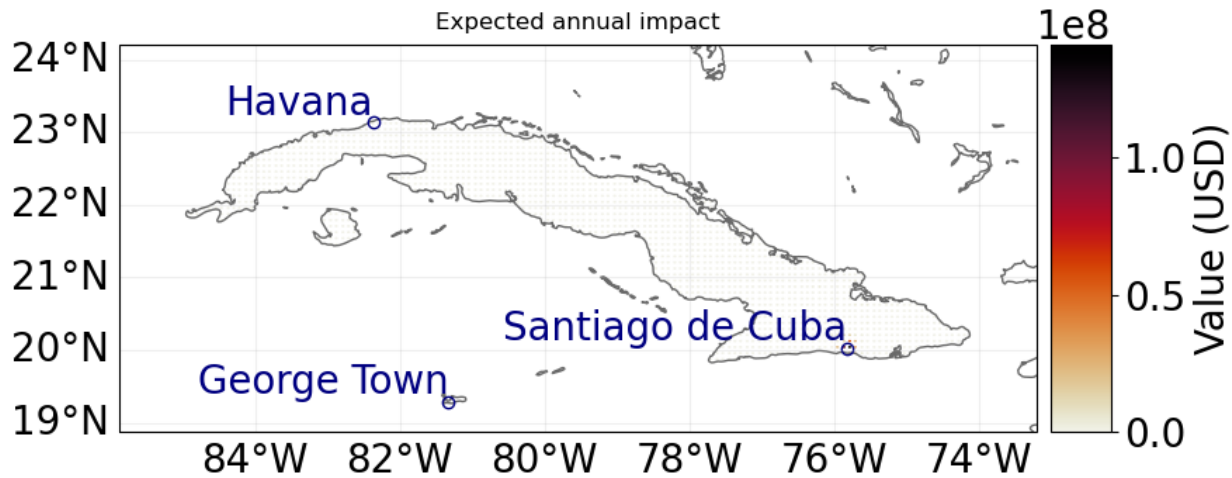
For example we can now obtain the aggregated average annual impact or plot the average annual impact in each exposure

location.

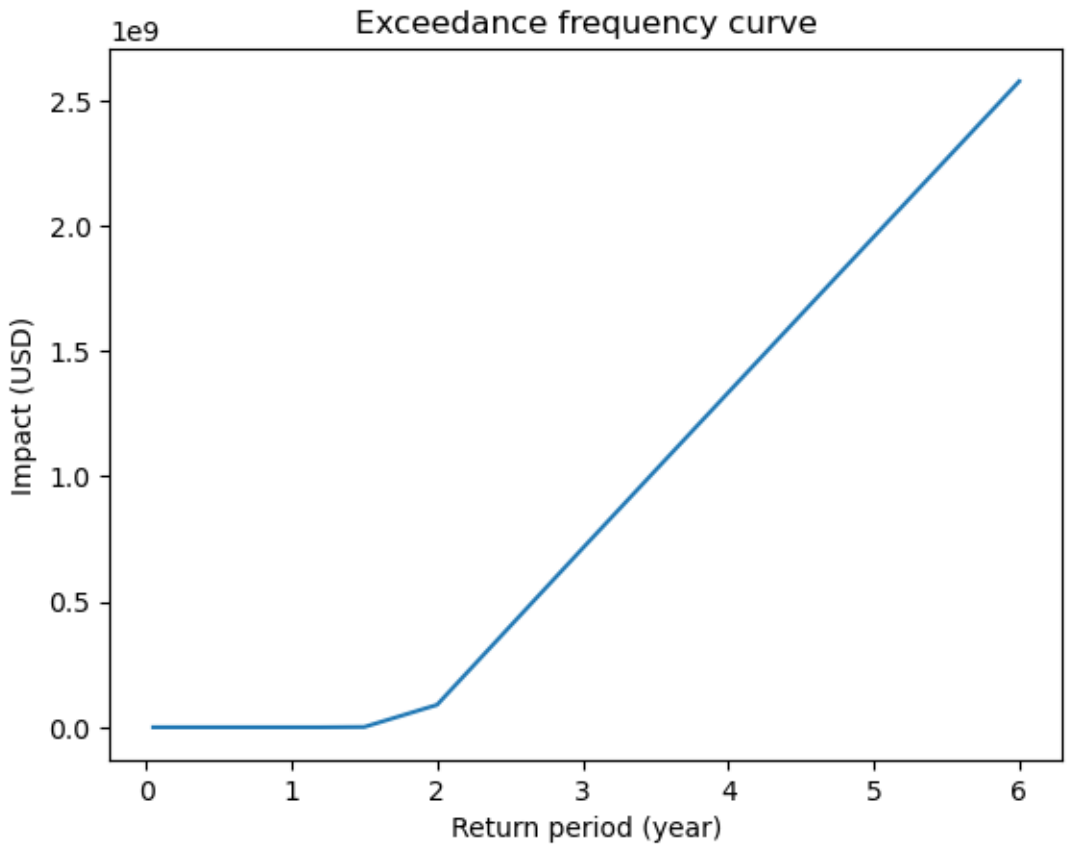
```
print(f"Aggregated average annual impact: {round(imp.aai_agg,0)} $")
```

```
Aggregated average annual impact: 563366225.0 $
```

```
imp.plot_hexbin_eai_exposure(buffer=1);
```




```
# Compute exceedance frequency curve
freq_curve = imp.calc_freq_curve()
freq_curve.plot();
```



Impact concatenation

There can be cases in which an impact function for a given hazard type is not constant throughout the year. This is for example the case in the context of agriculture. For example, if a crop is already harvested the impact of a certain weather event can be much lower or even zero. For such situations of two or more different impact functions for the same hazard and exposure type, it can be useful to split the events into subsets and compute impacts separately. In order to then analyze the total impact, the different impact subsets can be concatenated using the `Impact.concat` method. This is done here for the hypothetical example using LitPop as exposure and TCs as hazard. For illustration purposes, we misuse the LitPop exposure in this case as exposure of a certain crop. We assume a constant harvest day (17 October) after which the impact function is reduced by a factor of 10.

First, we prepare the hazard subsets.

```
from datetime import datetime, date
import pandas as pd

# set a harvest date
harvest_DOY = 290  # 17 October

# loop over all events an check if they happened before or after harvest
event_ids_post_harvest = []
event_ids_pre_harvest = []
for event_id in tc.event_id:
    event_date = tc.date[np.where(tc.event_id == event_id)[0][0]]
    day_of_year = (
```

(continues on next page)

(continued from previous page)

```
        event_date - date(datetime.fromordinal(event_date).year, 1, 1).toordinal() + 1
    )

    if day_of_year > harvest_DOY:
        event_ids_post_harvest.append(event_id)
    else:
        event_ids_pre_harvest.append(event_id)

tc_post_harvest = tc.select(event_id=event_ids_post_harvest)
tc_pre_harvest = tc.select(event_id=event_ids_pre_harvest)
# print('pre-harvest:', tc_pre_harvest.event_name)
# print('post-harvest:', tc_post_harvest.event_name)
```

Now we get two different impact functions, one valid for the exposed crop before harvest and one after harvest. Then, we compute the impacts for both phases separately.

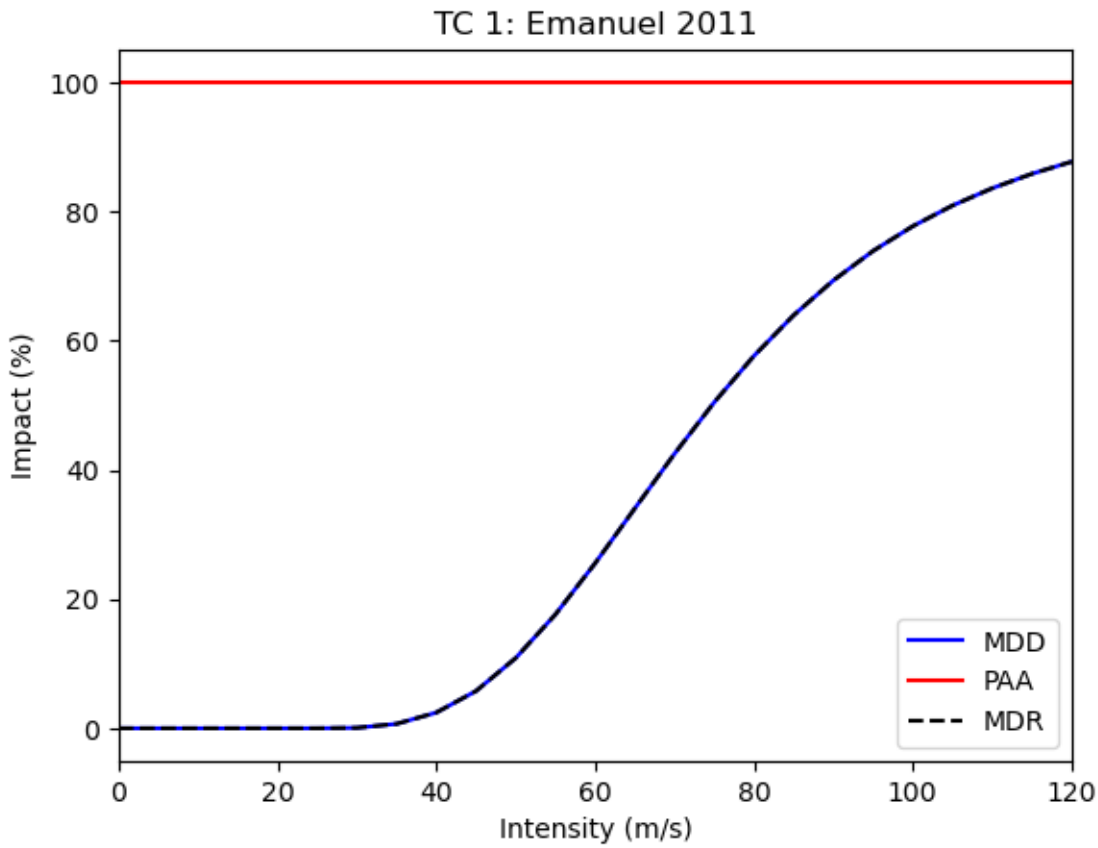
```
from climada.engine import Impact

# impact function TC
impf_tc = ImpfTropCyclone.from_emanuel_usa()
# impact function TC after harvest is by factor 0.5 smaller
impf_tc_posth = ImpfTropCyclone.from_emanuel_usa()
impf_tc_posth.mdd = impf_tc.mdd * 0.1
# add the impact function to an Impact function set
impf_set = ImpactFuncSet([impf_tc])
impf_set_posth = ImpactFuncSet([impf_tc_posth])
impf_set.check()
impf_set_posth.check()

# plot
impf_set.plot()
impf_set_posth.plot()

# Compute impacts
imp_preh = ImpactCalc(exp_lp, impf_set, tc_pre_harvest).impact(save_mat=True)
imp_posth = ImpactCalc(exp_lp, impf_set_posth, tc_post_harvest).impact(save_mat=True)
```

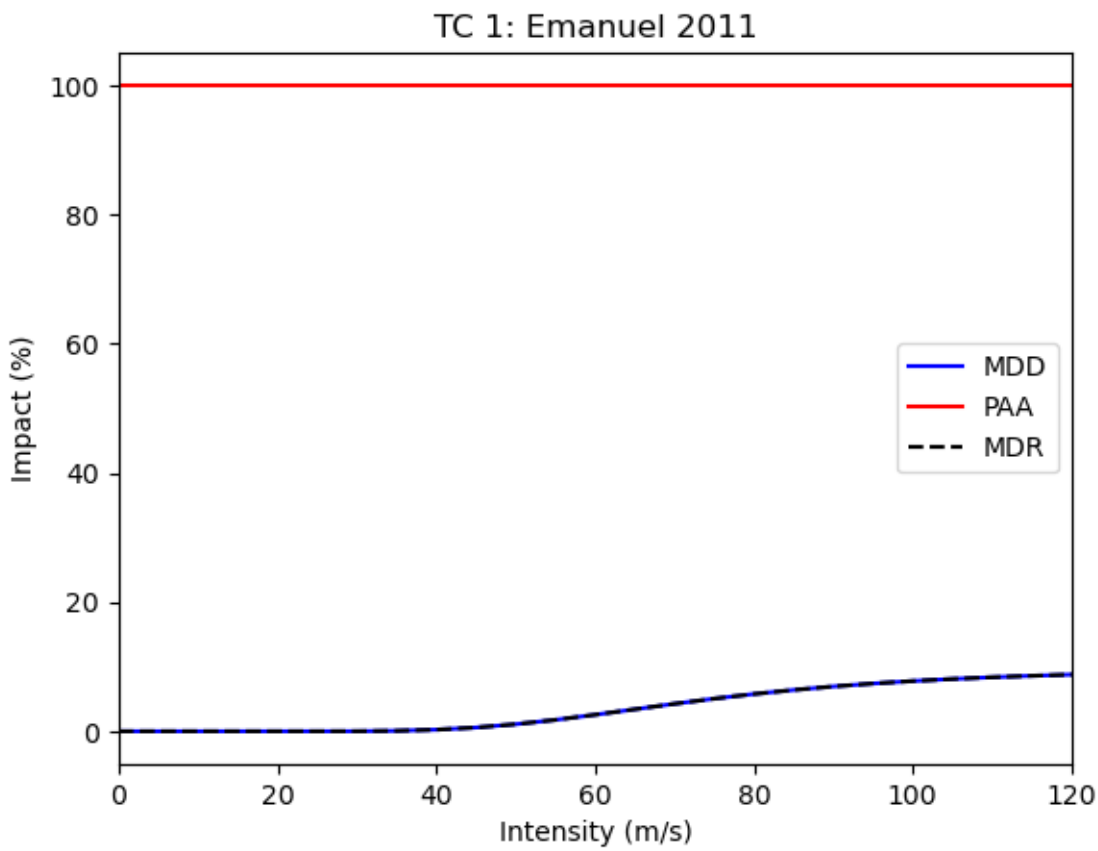
```
2023-01-26 11:58:52,364 - climada.entity.exposures.base - INFO - Exposures matching␣
↪centroids already found for TC
2023-01-26 11:58:52,364 - climada.entity.exposures.base - INFO - Existing centroids␣
↪will be overwritten for TC
2023-01-26 11:58:52,364 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 1388 centroids.
2023-01-26 11:58:52,370 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪4164 assets (>0) and 106 events.
2023-01-26 11:58:52,379 - climada.entity.exposures.base - INFO - Exposures matching␣
↪centroids already found for TC
2023-01-26 11:58:52,379 - climada.entity.exposures.base - INFO - Existing centroids␣
↪will be overwritten for TC
2023-01-26 11:58:52,382 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 1388 centroids.
2023-01-26 11:58:52,388 - climada.engine.impact_calc - INFO - Calculating impact for␣
```

(continues on next page)

(continued from previous page)

```
↪4164 assets (>0) and 14 events.
```

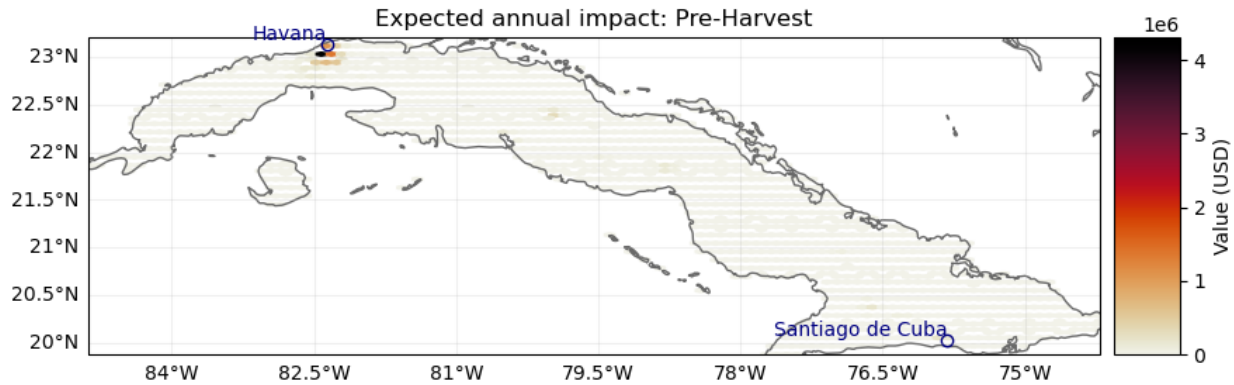


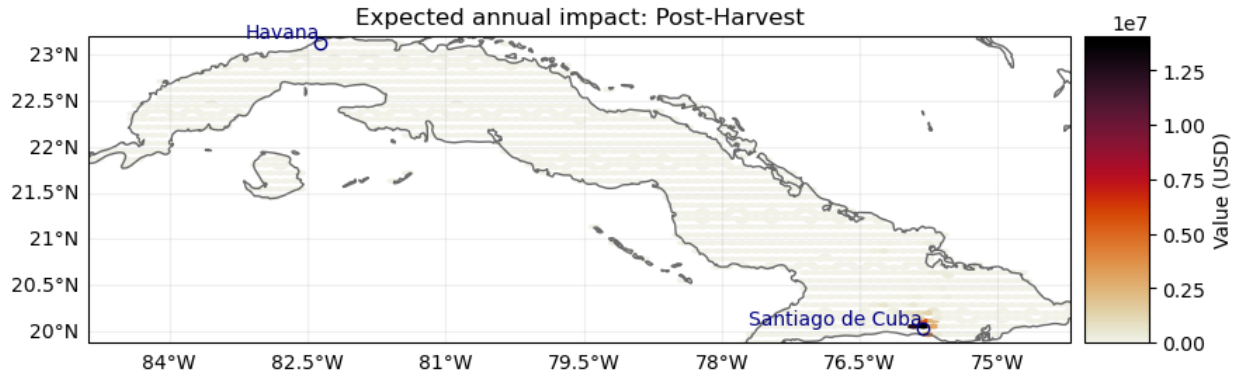


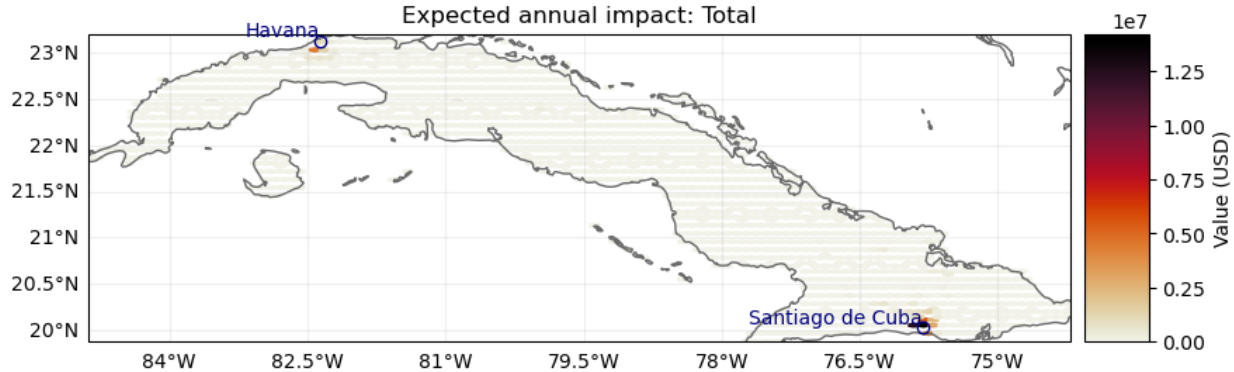
Now, we can concatenate the impacts again and plot the results

```
# Concatenate impacts again
imp_tot = Impact.concat([imp_preh, imp_posth])

# plot result
import matplotlib.pyplot as plt

ax = imp_preh.plot_hexbin_eai_exposure(gridsize=100, adapt_fontsize=False)
ax.set_title("Expected annual impact: Pre-Harvest")
ax = imp_posth.plot_hexbin_eai_exposure(gridsize=100, adapt_fontsize=False)
ax.set_title("Expected annual impact: Post-Harvest")
ax = imp_tot.plot_hexbin_eai_exposure(gridsize=100, adapt_fontsize=False)
ax.set_title("Expected annual impact: Total")
```

```
Text(0.5, 1.0, 'Expected annual impact: Total')
```







Quick examples - points, raster, custom

User defined point exposure and Tropical Cyclone hazard

```
%matplotlib inline
# EXAMPLE: POINT EXPOSURES WITH POINT HAZARD
import numpy as np
from climada.entity import Exposures, ImpactFuncSet, IFTropCyclone
from climada.hazard import Centroids, TCTracks, TropCyclone
from climada.engine import ImpactCalc

# Set Exposures in points
```

(continues on next page)

(continued from previous page)

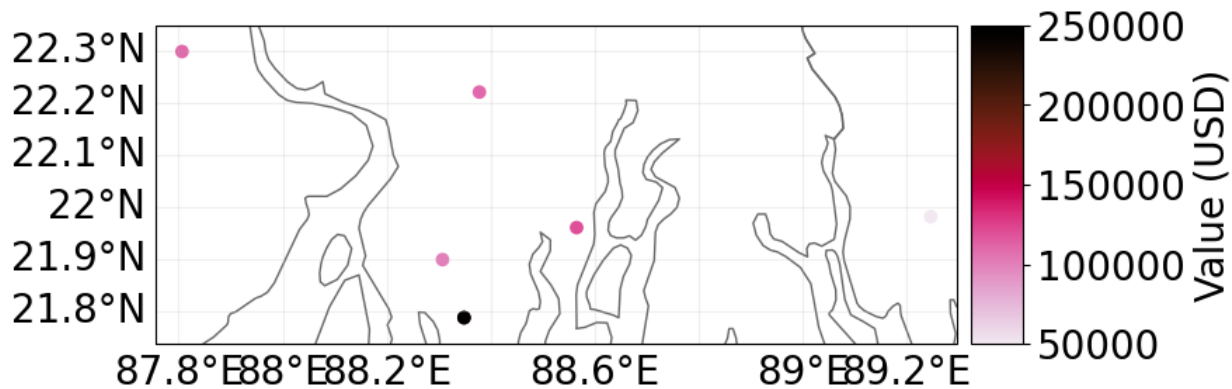
```
exp_pnt = Exposures(crs="epsg:4326")  # set coordinate system
exp_pnt.gdf["latitude"] = np.array(
    [21.899326, 21.960728, 22.220574, 22.298390, 21.787977, 21.787977, 21.981732]
)
exp_pnt.gdf["longitude"] = np.array(
    [88.307422, 88.565362, 88.378337, 87.806356, 88.348835, 88.348835, 89.246521]
)
exp_pnt.gdf["value"] = np.array([1.0e5, 1.2e5, 1.1e5, 1.1e5, 2.0e5, 2.5e5, 0.5e5])
exp_pnt.check()
exp_pnt.plot_scatter(buffer=0.05)

# Set Hazard in Exposures points
# set centroids from exposures coordinates
centr_pnt = Centroids.from_lat_lon(exp_pnt.latitude, exp_pnt.longitude, exp_pnt.crs)
# compute Hazard in that centroids
tr_pnt = TCTracks.from_ibtracs_netcdf(storm_id="2007314N10093")
tc_pnt = TropCyclone.from_tracks(tr_pnt, centroids=centr_pnt)
tc_pnt.check()
ax_pnt = tc_pnt.centroids.plot(
    c=np.array(tc_pnt.intensity[0, :].todense()).squeeze()
)  # plot intensity per point
ax_pnt.get_figure().colorbar(
    ax_pnt.collections[0], fraction=0.0175, pad=0.02
).set_label(
    "Intensity (m/s)"
)  # add colorbar

# Set impact function
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_pnt = ImpactFuncSet([impf_tc])
impf_pnt.check()

# Get the hazard type and hazard id
[haz_type] = impf_set.get_hazard_types()
[haz_id] = impf_set.get_ids()[haz_type]
# Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + haz_type}, inplace=True)
exp_lp.gdf["impf_" + haz_type] = haz_id
exp_lp.gdf.head()

# Compute Impact
imp_pnt = ImpactCalc(exp_pnt, impf_pnt, tc_pnt).impact()
# nearest neighbor of exposures to centroids gives identity
print(
    "Nearest neighbor hazard.centroids indexes for each exposure:",
    exp_pnt.gdf["centr_TC"].values,
)
imp_pnt.plot_scatter_eai_exposure(ignore_zero=False, buffer=0.05);
```

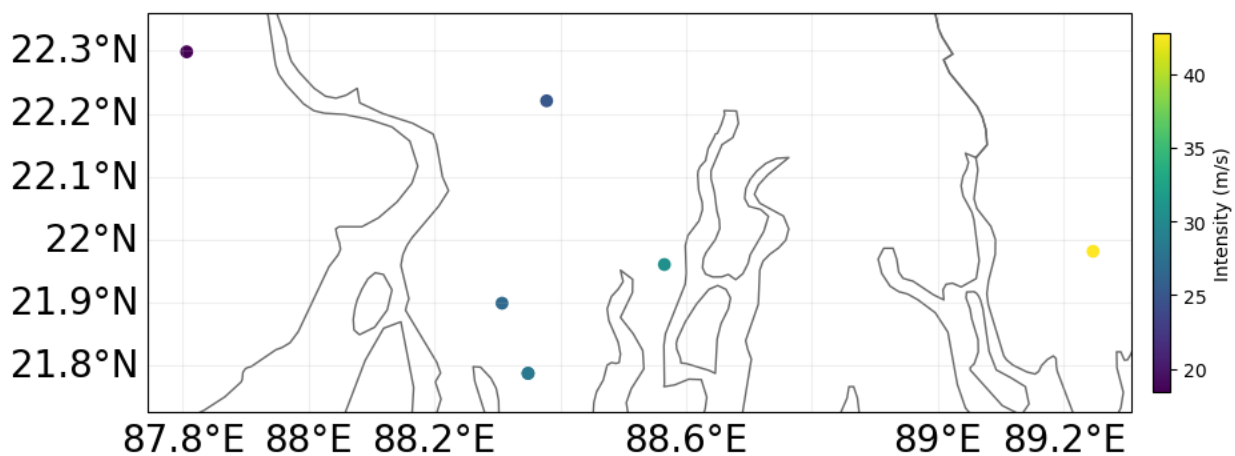
```
2023-01-26 11:59:01,455 - climada.entity.exposures.base - INFO - Setting impf_ to␣
↪default impact functions ids 1.
2023-01-26 11:59:01,457 - climada.entity.exposures.base - INFO - category_id not set.
```

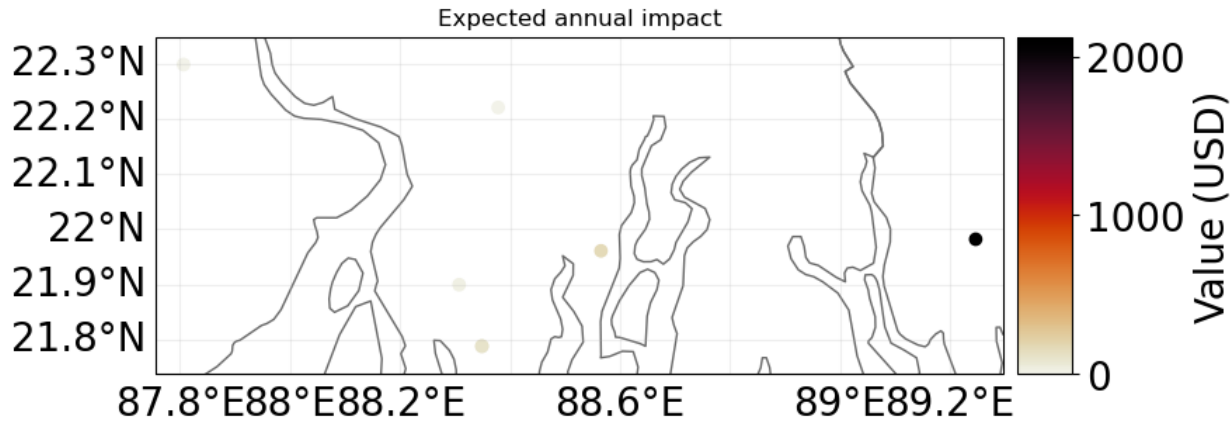
(continues on next page)

(continued from previous page)

```
2023-01-26 11:59:01,458 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:01,460 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:01,463 - climada.entity.exposures.base - INFO - geometry not set.
2023-01-26 11:59:01,464 - climada.entity.exposures.base - INFO - region_id not set.
2023-01-26 11:59:01,466 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:03,801 - climada.hazard.tc_tracks - INFO - Progress: 100%
2023-01-26 11:59:03,846 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2023-01-26 11:59:04,466 - climada.util.coordinates - INFO - dist_to_coast: UTM 32645␣
↪(1/1)
2023-01-26 11:59:04,580 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to 7␣
↪coastal centroids.
2023-01-26 11:59:04,595 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 11:59:05,457 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 11:59:05,458 - climada.entity.exposures.base - INFO - Matching 7 exposures␣
↪with 7 centroids.
2023-01-26 11:59:05,463 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪21 assets (>0) and 1 events.
Nearest neighbor hazard.centroids indexes for each exposure: [0 1 2 3 4 5 6]
```







Raster from file

```
# EXAMPLE: RASTER EXPOSURES WITH RASTER HAZARD
from rasterio.warp import Resampling
from climada.entity import LitPop, ImpactFuncSet, ImpactFunc
from climada.hazard import Hazard
from climada.engine import Impact
from climada.util.constants import HAZ_DEMO_FL

# Exposures belonging to a raster (the raser information is contained in the meta␣
↪attribute)
exp_ras = LitPop.from_countries(
    countries=["VEN"], res_arcsec=300, fin_mode="income_group"
)
exp_ras.gdf.reset_index()
exp_ras.check()
exp_ras.plot_raster()
print("\n Raster properties exposures:", exp_ras.meta)

# Initialize hazard object with haz_type = 'FL' (for Flood)
hazard_type = "FL"
# Load a previously generated (either with CLIMADA or other means) hazard
# from file (HAZ_DEMO_FL) and resample the hazard raster to the exposures' ones
# Hint: check how other resampling methods affect to final impact
haz_ras = Hazard.from_raster(
    [HAZ_DEMO_FL],
    haz_type=hazard_type,
    dst_crs=exp_ras.meta["crs"],
    transform=exp_ras.meta["transform"],
    width=exp_ras.meta["width"],
    height=exp_ras.meta["height"],
    resampling=Resampling.nearest,
)
haz_ras.intensity[haz_ras.intensity == -9999] = 0  # correct no data values
haz_ras.check()
haz_ras.plot_intensity(1)
print("Raster properties centroids:", haz_ras.centroids.meta)
```

(continues on next page)

(continued from previous page)

```
# Set dummy impact function
intensity = np.linspace(0, 10, 100)
mdd = np.linspace(0, 10, 100)
paa = np.ones(intensity.size)
impf_dum = ImpactFunc(hazard_type, haz_id, intensity, mdd, paa, "m", "dummy")
# Add the impact function to the impact function set
impf_ras = ImpactFuncSet([impf_dum])
impf_ras.check()

# Exposures: rename column and assign id
exp_lp.gdf.rename(columns={"impf_": "impf_" + hazard_type}, inplace=True)
exp_lp.gdf["impf_" + haz_type] = haz_id
exp_lp.gdf.head()

# Compute impact
imp_ras = ImpactCalc(exp_ras, impf_ras, haz_ras).impact(save_mat=False)
# nearest neighbor of exposures to centroids is not identity because litpop does not␣
↪contain data outside the country polygon
print(
    "\n Nearest neighbor hazard.centroids indexes for each exposure:",
    exp_ras.gdf["centr_FL"].values,
)
imp_ras.plot_raster_eai_exposure();
```

```
2023-01-26 11:59:11,285 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: VEN (862)...

2023-01-26 11:59:13,749 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,751 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:13,783 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:13,785 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,787 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:13,850 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,852 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:13,926 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,928 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:13,984 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:13,985 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,045 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,046 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2023-01-26 11:59:14,105 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,108 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,167 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,168 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,220 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,221 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,252 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,253 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,255 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,283 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,284 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,286 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,314 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,315 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,317 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,348 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,349 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,351 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,381 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,383 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,386 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,416 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:14,417 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,419 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,491 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,493 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

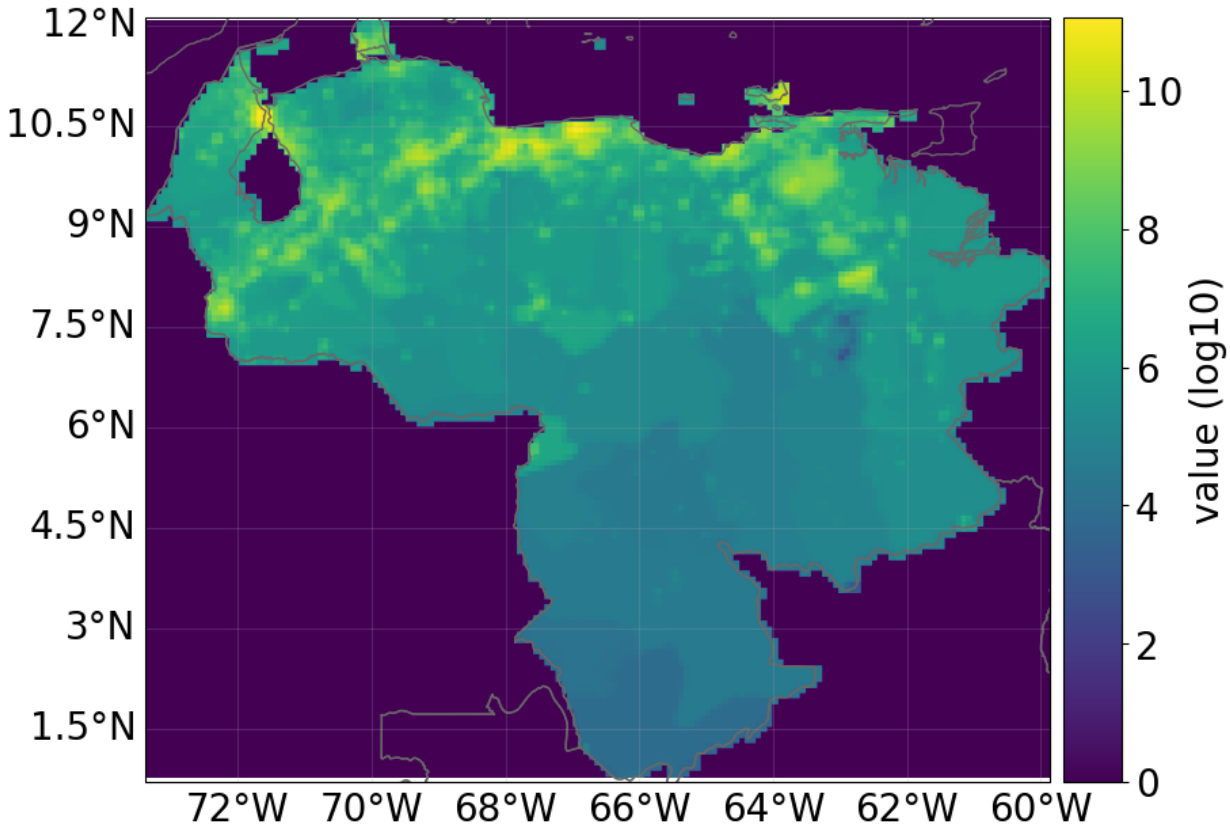
```
↪Version v4.11
2023-01-26 11:59:14,556 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,557 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,622 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,624 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,703 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,705 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,779 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,780 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,854 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,855 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,922 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,924 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:14,988 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:14,990 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,026 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:15,028 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:15,029 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,065 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:15,066 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:15,069 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,130 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:15,131 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,200 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
2023-01-26 11:59:15,202 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,263 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2018. Using nearest available year for GPW data: 2020
```

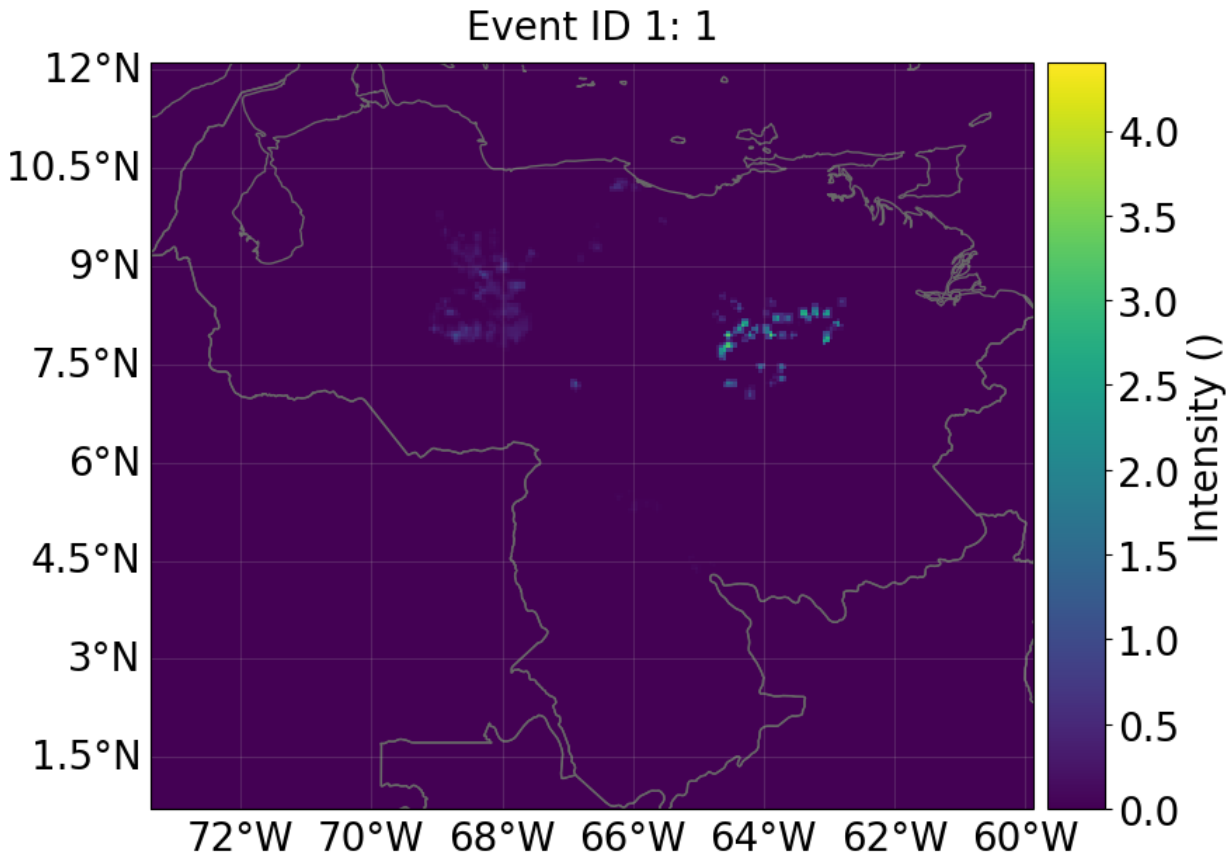
```
2023-01-26 11:59:15,265 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2023-01-26 11:59:15,289 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2023-01-26 11:59:15,833 - climada.util.finance - INFO - GDP VEN 2014: 4.824e+11.
2023-01-26 11:59:15,909 - climada.util.finance - INFO - Income group VEN 2018: 3.
2023-01-26 11:59:15,933 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:59:15,934 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:15,936 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:15,937 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:15,939 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:15,949 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:59:15,951 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:15,952 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:15,953 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:15,955 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:15,962 - climada.util.coordinates - INFO - Raster from resolution 0.
↪08333332999999987 to 0.08333332999999987.

 Raster properties exposures: {'width': 163, 'height': 138, 'crs': <Geographic 2D␣
↪CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
, 'transform': Affine(0.08333333000000209, 0.0, -73.41666666500001,
       0.0, -0.08333332999999987, 12.166666665)}
2023-01-26 11:59:23,374 - climada.util.coordinates - INFO - Reading C:\Users\
↪F80840370\climada\demo\data\SC22000_VE__M1.grd.gz
2023-01-26 11:59:25,519 - climada.util.coordinates - INFO - Reading C:\Users\
↪F80840370\climada\demo\data\SC22000_VE__M1.grd.gz
Raster properties centroids: {'driver': 'GSBG', 'dtype': 'float32', 'nodata': 1.
↪701410009187828e+38, 'width': 163, 'height': 138, 'count': 1, 'crs': <Geographic 2D␣
↪CRS: EPSG:4326>
Name: WGS 84
Axis Info [ellipsoidal]:
- Lat[north]: Geodetic latitude (degree)
- Lon[east]: Geodetic longitude (degree)
Area of Use:
- undefined
Datum: World Geodetic System 1984
- Ellipsoid: WGS 84
- Prime Meridian: Greenwich
, 'transform': Affine(0.08333333000000209, 0.0, -73.41666666500001,
       0.0, -0.08333332999999987, 12.166666665)}
2023-01-26 11:59:28,695 - climada.entity.exposures.base - INFO - No specific impact␣
```

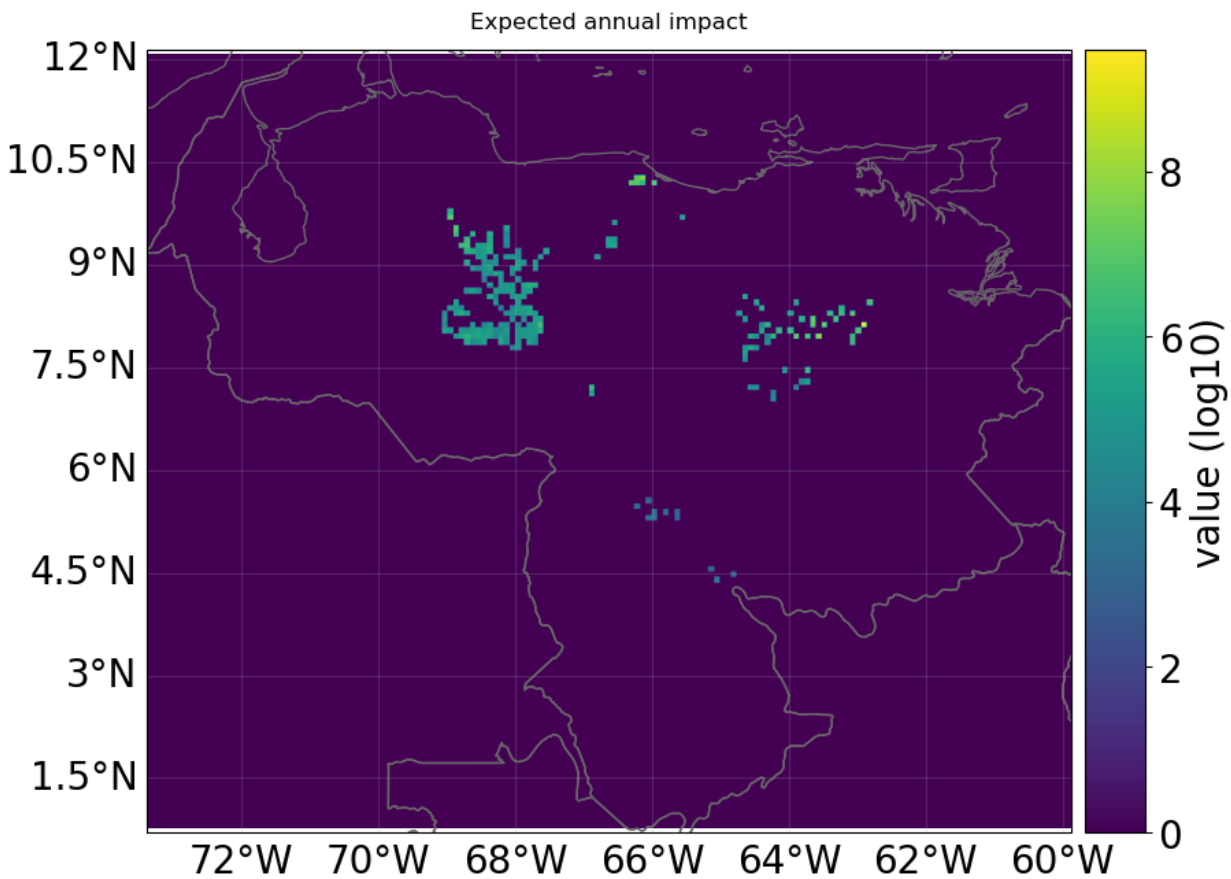
(continues on next page)

(continued from previous page)

```
↪function column found for hazard FL. Using the anonymous 'impf_' column.
2023-01-26 11:59:28,695 - climada.entity.exposures.base - INFO - Matching 10772␣
↪exposures with 22494 centroids.
2023-01-26 11:59:28,704 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪32310 assets (>0) and 1 events.

 Nearest neighbor hazard.centroids indexes for each exposure: [  39   40   41 ...␣
↪3387 3551 2721]
2023-01-26 11:59:28,714 - climada.util.coordinates - INFO - Raster from resolution 0.
↪08333332999999987 to 0.08333332999999987.
```




Event ID 1: 1
12°N
10.5°N
9°N
7.5°N
6°N
4.5°N
3°N
1.5°N
72°W
70°W
68°W
66°W
64°W
62°W
60°W
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
Intensity ()



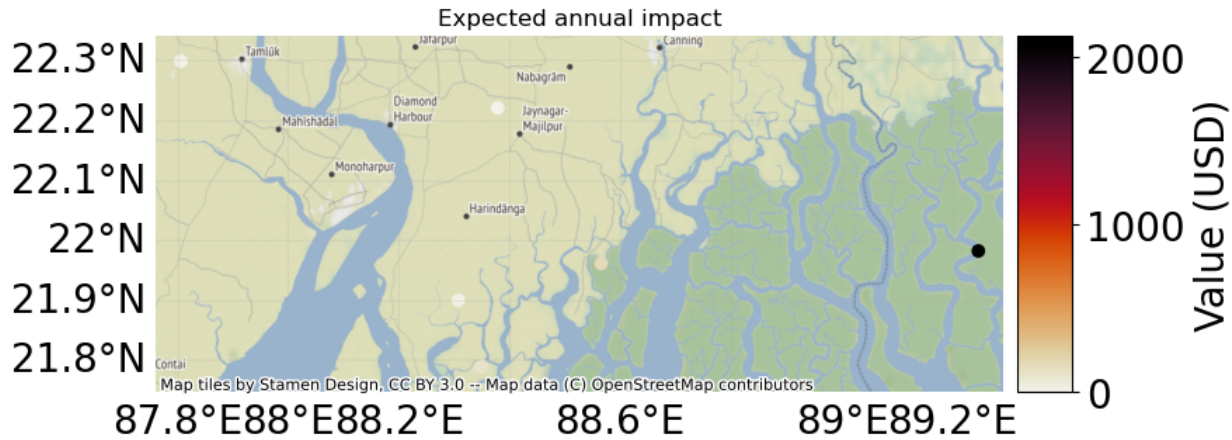
Visualization

Making plots

The expected annual impact per exposure can be visualized through different methods: `plot_hexbin_eai_exposure()`, `plot_scatter_eai_exposur()`, `plot_raster_eai_exposure()` and `plot_basemap_eai_exposure()` (similarly as with `Exposures`).

```
imp_pnt.plot_basemap_eai_exposure(buffer=5000);
```

```
2023-01-26 11:59:40,355 - climada.util.coordinates - INFO - Setting geometry points.
2023-01-26 11:59:40,364 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
2023-01-26 11:59:43,294 - climada.entity.exposures.base - INFO - Setting latitude and␣
↪longitude attributes.
```



Making videos

Given a fixed exposure and impact functions, a sequence of hazards can be visualized hitting the exposures.

```
# exposure
from climada.entity import add_sea
from climada_petals.entity import BlackMarble

exp_video = BlackMarble()
exp_video.set_countries(["Cuba"], 2016, res_km=2.5)
exp_video.check()

# impact function
impf_def = ImpfTropCyclone.from_emanuel_usa()
impfs_video = ImpactFuncSet([impf_def])
impfs_video.check()

# compute sequence of hazards using TropCyclone video_intensity method
exp_sea = add_sea(exp_video, (100, 5))
centr_video = Centroids.from_lat_lon(exp_sea.latitude, exp_sea.longitude)
centr_video.check()

track_name = "2017242N16333"
tr_irma = TCTracks.from_ibtracs_netcdf(provider="usa", storm_id=track_name)  # IRMA␣
↪2017

tc_video = TropCyclone()
tc_list, _ = tc_video.video_intensity(
    track_name, tr_irma, centr_video
)  # empty file name to not to write the video

# generate video of impacts
file_name = "./results/irma_imp_fl.gif"
imp_video = Impact()
imp_list = imp_video.video_direct_impact(exp_video, impfs_video, tc_list, file_name)
```

```
2023-01-26 11:59:44,414 - climada.util.finance - INFO - GDP CUB 2016: 9.137e+10.
2023-01-26 11:59:44,483 - climada.util.finance - INFO - Income group CUB 2016: 3.
```

(continues on next page)

(continued from previous page)

```
2023-01-26 11:59:44,483 - climada_petals.entity.exposures.black_marble - INFO -␣
↪Nightlights from NASA's earth observatory for year 2016.
2023-01-26 11:59:48,126 - climada_petals.entity.exposures.black_marble - INFO -␣
↪Processing country Cuba.
2023-01-26 11:59:48,987 - climada_petals.entity.exposures.black_marble - INFO -␣
↪Generating resolution of approx 2.5 km.
2023-01-26 11:59:49,129 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - geometry not set.
2023-01-26 11:59:49,137 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:49,159 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2023-01-26 11:59:49,159 - climada.entity.exposures.base - INFO - category_id not set.
2023-01-26 11:59:49,159 - climada.entity.exposures.base - INFO - cover not set.
2023-01-26 11:59:49,167 - climada.entity.exposures.base - INFO - deductible not set.
2023-01-26 11:59:49,167 - climada.entity.exposures.base - INFO - geometry not set.
2023-01-26 11:59:49,167 - climada.entity.exposures.base - INFO - centr_ not set.
2023-01-26 11:59:49,167 - climada.entity.exposures.base - INFO - Adding sea at 5 km␣
↪resolution and 100 km distance from coast.
2023-01-26 11:59:50,567 - climada.hazard.tc_tracks - INFO - Progress: 100%
2023-01-26 11:59:50,589 - climada.hazard.centroids.centr - INFO - Convert centroids␣
↪to GeoSeries of Point shapes.
2023-01-26 12:00:02,901 - climada.util.coordinates - INFO - dist_to_coast: UTM 32616␣
↪(1/3)
2023-01-26 12:00:09,800 - climada.util.coordinates - INFO - dist_to_coast: UTM 32617␣
↪(2/3)
2023-01-26 12:00:32,928 - climada.util.coordinates - INFO - dist_to_coast: UTM 32618␣
↪(3/3)
2023-01-26 12:00:42,661 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪23083 coastal centroids.
2023-01-26 12:00:42,684 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,702 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪26071 coastal centroids.
2023-01-26 12:00:42,733 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,750 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪29579 coastal centroids.
2023-01-26 12:00:42,781 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,818 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪32525 coastal centroids.
2023-01-26 12:00:42,849 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,865 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪35241 coastal centroids.
2023-01-26 12:00:42,902 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:42,934 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪37535 coastal centroids.
2023-01-26 12:00:42,981 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,002 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪39661 coastal centroids.
2023-01-26 12:00:43,034 - climada.hazard.trop_cyclone - INFO - Progress: 100%
```

(continues on next page)

(continued from previous page)

```
2023-01-26 12:00:43,065 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪42459 coastal centroids.
2023-01-26 12:00:43,103 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,118 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪43981 coastal centroids.
2023-01-26 12:00:43,181 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,203 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪45805 coastal centroids.
2023-01-26 12:00:43,250 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,266 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪47236 coastal centroids.
2023-01-26 12:00:43,319 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,334 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪48530 coastal centroids.
2023-01-26 12:00:43,381 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,404 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪48252 coastal centroids.
2023-01-26 12:00:43,435 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,466 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪47331 coastal centroids.
2023-01-26 12:00:43,504 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,535 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪45981 coastal centroids.
2023-01-26 12:00:43,582 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,604 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪44618 coastal centroids.
2023-01-26 12:00:43,635 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,667 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪43393 coastal centroids.
2023-01-26 12:00:43,704 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,720 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪42463 coastal centroids.
2023-01-26 12:00:43,767 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,782 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪41702 coastal centroids.
2023-01-26 12:00:43,820 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,836 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪41057 coastal centroids.
2023-01-26 12:00:43,869 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,900 - climada.hazard.trop_cyclone - INFO - Mapping 1 tracks to␣
↪40802 coastal centroids.
2023-01-26 12:00:43,936 - climada.hazard.trop_cyclone - INFO - Progress: 100%
2023-01-26 12:00:43,951 - climada.entity.exposures.base - INFO - Matching 21923␣
↪exposures with 49817 centroids.
2023-01-26 12:00:43,951 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2023-01-26 12:00:44,005 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,005 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,021 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
```

(continues on next page)

(continued from previous page)

```
2023-01-26 12:00:44,021 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,021 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,036 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,052 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,052 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,052 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,067 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,067 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,067 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,084 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,084 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
```

```
2023-01-26 12:00:44,084 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,105 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,105 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,105 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,121 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,121 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,136 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,136 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,152 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,152 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,167 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,167 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,167 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,183 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,183 - climada.entity.exposures.base - INFO - No specific impact␣
```

(continues on next page)

(continued from previous page)

```
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,199 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,205 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,205 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,205 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,221 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,221 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,221 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,237 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,237 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,237 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,252 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,252 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,252 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,268 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,268 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,268 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,283 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,283 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,283 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,299 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,306 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,306 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,321 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,321 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,321 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,337 - climada.engine.impact - WARNING - The use of Impact().
```

(continues on next page)

(continued from previous page)

```
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,337 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,337 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,352 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,352 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,352 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,368 - climada.engine.impact - WARNING - The use of Impact().
↪calc() is deprecated. Use ImpactCalc().impact() instead.
2023-01-26 12:00:44,368 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2023-01-26 12:00:44,368 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪43962 assets (>0) and 1 events.
2023-01-26 12:00:44,384 - climada.engine.impact - INFO - Generating video ./results/
↪irma_imp_fl.gif
```

```
22it [09:44, 26.57s/it]
```

2.5.2 Impact Functions

What is an impact function?

An impact function relates the percentage of damage in the exposure to the hazard intensity, also commonly referred to as a "vulnerability curve" in the modelling community. Every hazard and exposure types are characterized by an impact function.

What is the difference between `ImpactFunc` and `ImpactFuncSet`?

An `ImpactFunc` is a class for a single impact function. E.g. a function that relates the percentage of damage of a reinforced concrete building (exposure) to the wind speed of a tropical cyclone (hazard intensity).

An `ImpactFuncSet` class is a container that contains multiple `ImpactFunc`. For instance, there are 100 `ImpactFunc`

represent 100 types of buildings exposed to tropical cyclone's wind damage. These 100 `ImpactFunc` are all gathered in an `ImpactFuncSet`.

What does an `ImpactFunc` look like in CLIMADA?

The `ImpactFunc` class requires users to define the following attributes.

Mandatory attributes	Data Type	Description
haz_type	(str)	Hazard type acronym (e.g. 'TC')
id	(int or str)	Unique id of the impact function. Exposures of the same type will refer to the same impact function id
name	(str)	Name of the impact function
intensity	(np.array)	Intensity values
intensity_unit	(str)	Unit of the intensity
mdd	(np.array)	Mean damage (impact) degree for each intensity (numbers in [0,1])
paa	(np.array)	Percentage of affected assets (exposures) for each intensity (numbers in [0,1])

Users may use `ImpactFunc.check()` to check that the attributes have been set correctly. The mean damage ratio `mdr` (mdr=mdd*paa) is calculated by the method `ImpactFunc.calc_mdr()`.

What does an `ImpactFuncSet` look like in CLIMADA?

The `ImpactFuncSet` class contains all the `ImpactFunc` classes. Users are not required to define any attributes in `ImpactFuncSet`.

To add an `ImpactFunc` into an `ImpactFuncSet`, simply use the method `ImpactFuncSet.append(ImpactFunc)`. If the users only has one impact function, they should generate an `ImpactFuncSet` that contains one impact function. `ImpactFuncSet` is to be used in the *impact calculation*.

Attributes	Data Type	Description
_data	(dict)	Contains `ImpactFunc` classes. Not suppossed to be directly accessed. Use the class methods instead.

Part 1: Defining `ImpactFunc` from your own data

The essential attributes are listed in the table above. The following example shows you how to define an `ImpactFunc` from scratch, and using the method `ImpactFunc.calc_mdr()` to calculate the mean damage ratio.

Generate a dummy impact function from scratch.

Here we generate an impact function with random dummy data for illustrative reasons. Assuming this impact function is a function that relates building damage to tropical cyclone (TC) wind, with an arbitrary id 3.

```
import numpy as np
from climada.entity import ImpactFunc
```

(continues on next page)

(continued from previous page)

```
# We initialise a dummy ImpactFunc for tropical cyclone wind damage to building.
# Giving the ImpactFunc an arbitrary id 3.
haz_type = "TC"
id = 3
name = "TC building damage"
# provide unit of the hazard intensity
intensity_unit = "m/s"
# provide values for the hazard intensity, mdd, and paa
intensity = np.linspace(0, 100, num=15)
mdd = np.concatenate((np.array([0]), np.sort(np.random.rand(14))), axis=0)
paa = np.concatenate((np.array([0]), np.sort(np.random.rand(14))), axis=0)
imp_fun = ImpactFunc(
    id=id,
    name=name,
    intensity_unit=intensity_unit,
    haz_type=haz_type,
    intensity=intensity,
    mdd=mdd,
    paa=paa,
)

# check if the all the attributes are set correctly
imp_fun.check()
```

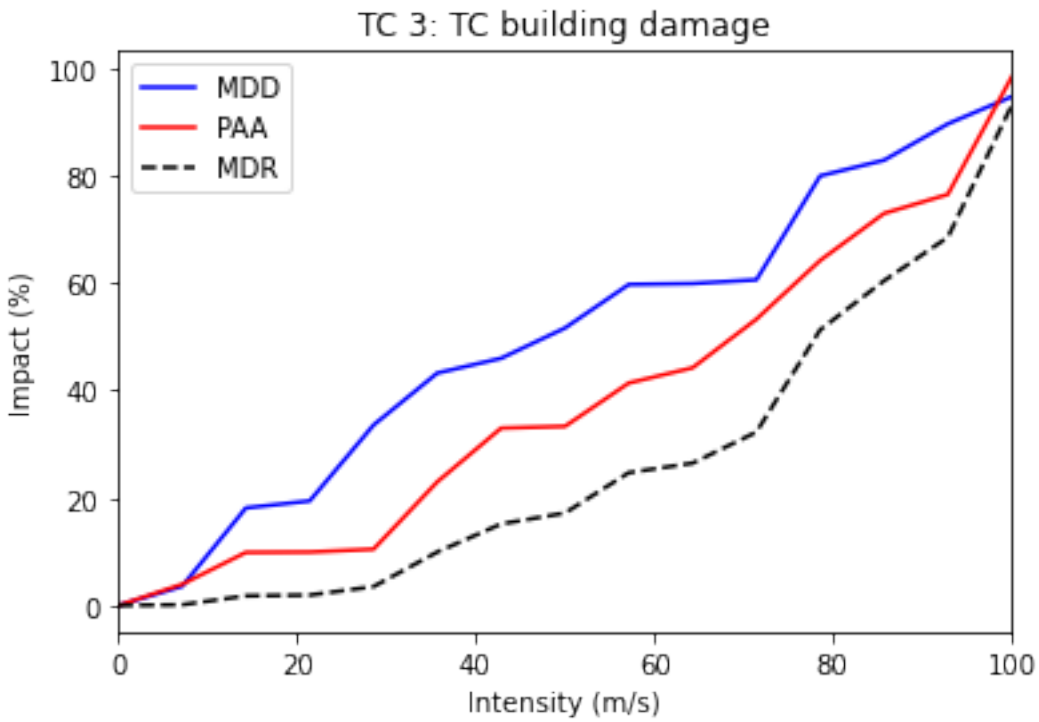
```
# Calculate the mdr at hazard intensity 18.7 m/s
print("Mean damage ratio at intensity 18.7 m/s: ", imp_fun.calc_mdr(18.7))
```

```
Mean damage ratio at intensity 18.7 m/s:  0.01878041941423081
```

Visualise the Impact function

The method `plot()` uses the matplotlib's axes plot function to visualise the impact function. It returns a figure and axes, which can be modified by users.

```
# plot impact function
imp_fun.plot();
```



Part 2: Loading impact functions from CLIMADA in-built impact functions

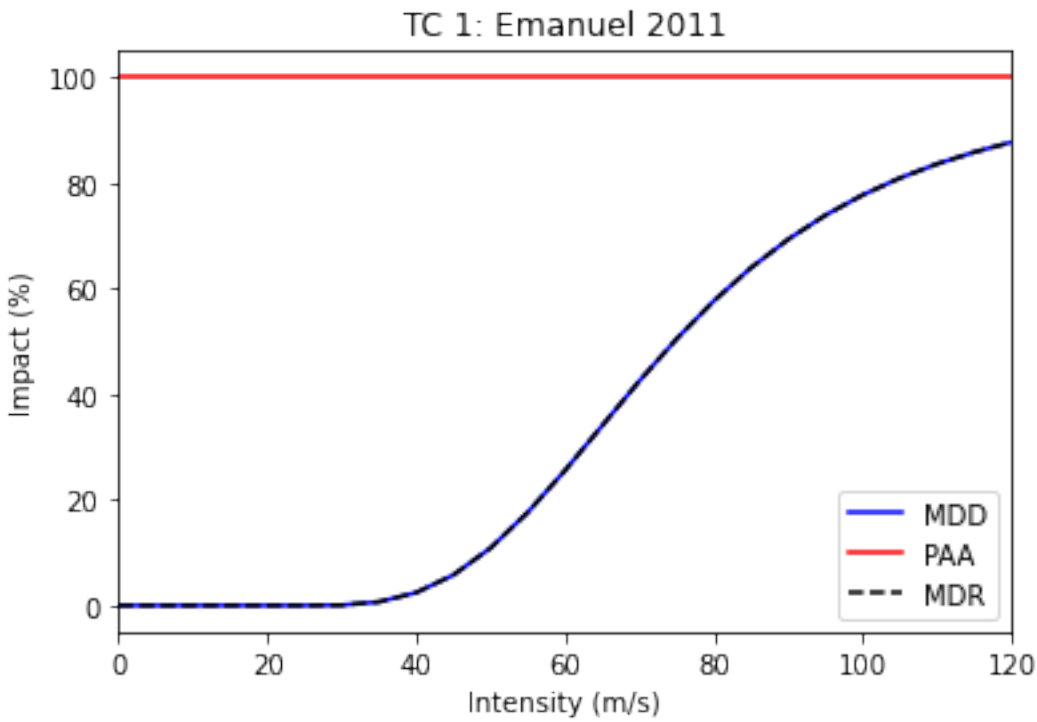
In CLIMADA there is several defined impact functions that users can directly load and use them. However, users should be aware of the applications of the impact functions to types of assets, reading the background references of the impact functions are strongly recommended. Currently available perils include tropical cyclones, river floods, European windstorm, crop yield, and drought.Continuous updates of perils are available. Here we use the impact function of tropical cyclones as an example.

Loading CLIMADA in-built impact function for tropical cyclones

`ImpfTropCyclone` is a derivated class of `ImpactFunc`. This in-built impact function estimates the insured property damages by tropical cyclone wind in USA, following the reference paper Emanuel (2011).

To generate this impact function, method `set_emanual_usa()` is used.

```
from climada.entity import ImpfTropCyclone

# Here we generate the impact function for TC damage using the formula of Emanuel 2011
impFunc_emanuel_usa = ImpfTropCyclone.from_emanuel_usa()
# plot the impact function
impFunc_emanuel_usa.plot();
```



Part 3: Add `ImpactFunc` into the container `ImpactFuncSet`

`ImpactFuncSet` is a container of multiple `ImpactFunc`, it is part of the arguments in `ImpactCalc.impact()` (see *the impact tutorial*).

Here we generate 2 arbitrary impact functions and add them into an `ImpactFuncSet` class. To add them into the container, simply use the method `ImpactFuncSet.append(ImpactFunc)`.

```
import numpy as np
import matplotlib.pyplot as plt
from climada.entity import ImpactFunc, ImpactFuncSet

# generate the 1st arbitrary impact function
haz_type = "TC"
id = 1
name = "TC Default Damage Function"
intensity_unit = "m/s"
intensity = np.linspace(0, 100, num=10)
mdd = np.concatenate((np.array([0]), np.sort(np.random.rand(9))), axis=0)
paa = np.concatenate((np.array([0]), np.sort(np.random.rand(9))), axis=0)
imp_fun_1 = ImpactFunc(
    id=id,
    name=name,
    intensity_unit=intensity_unit,
    haz_type=haz_type,
    intensity=intensity,
    mdd=mdd,
    paa=paa,
```

(continues on next page)

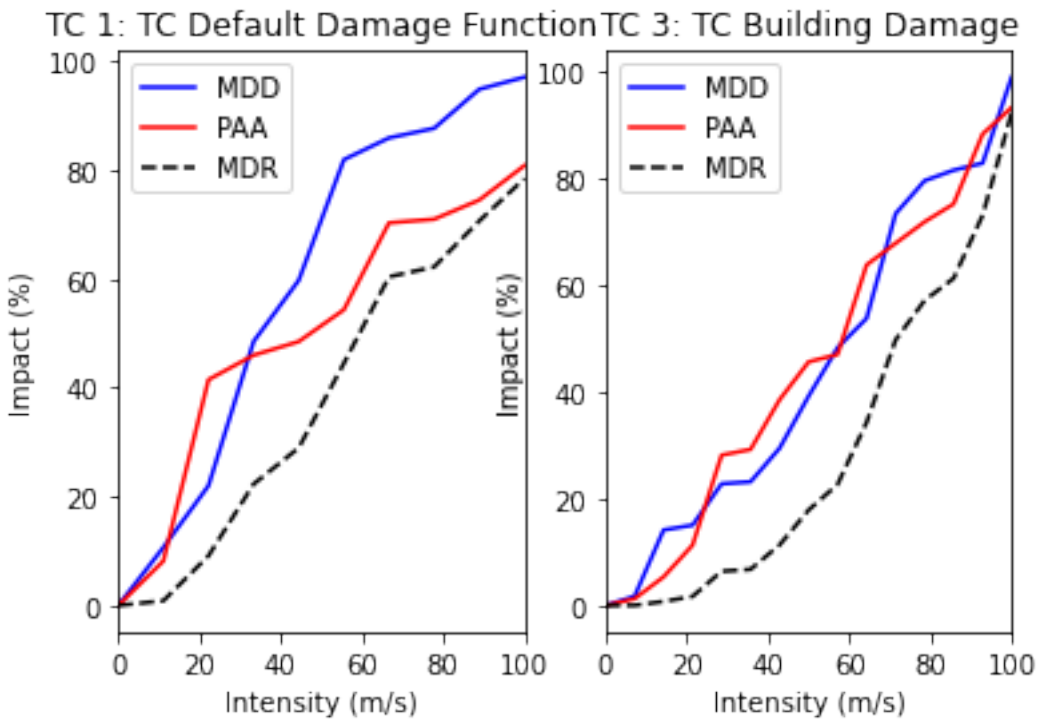
(continued from previous page)

```
)
imp_fun_1.check()

# generate the 2nd arbitrary impact function
haz_type = "TC"
id = 3
name = "TC Building Damage"
intensity_unit = "m/s"
intensity = np.linspace(0, 100, num=15)
mdd = np.concatenate((np.array([0]), np.sort(np.random.rand(14))), axis=0)
paa = np.concatenate((np.array([0]), np.sort(np.random.rand(14))), axis=0)
imp_fun_3 = ImpactFunc(
    id=id,
    name=name,
    intensity_unit=intensity_unit,
    haz_type=haz_type,
    intensity=intensity,
    mdd=mdd,
    paa=paa,
)
imp_fun_3.check()

# add the 2 impact functions into ImpactFuncSet
imp_fun_set = ImpactFuncSet([imp_fun_1, imp_fun_3])
```

Plotting all the impact functions in an `ImpactFuncSet`

The method `plot()` in `ImpactFuncSet` also uses the the matplotlib's axes plot function to visualise the impact functions, returning a figure with all the subplots of impact functions. Users may modify these plots.

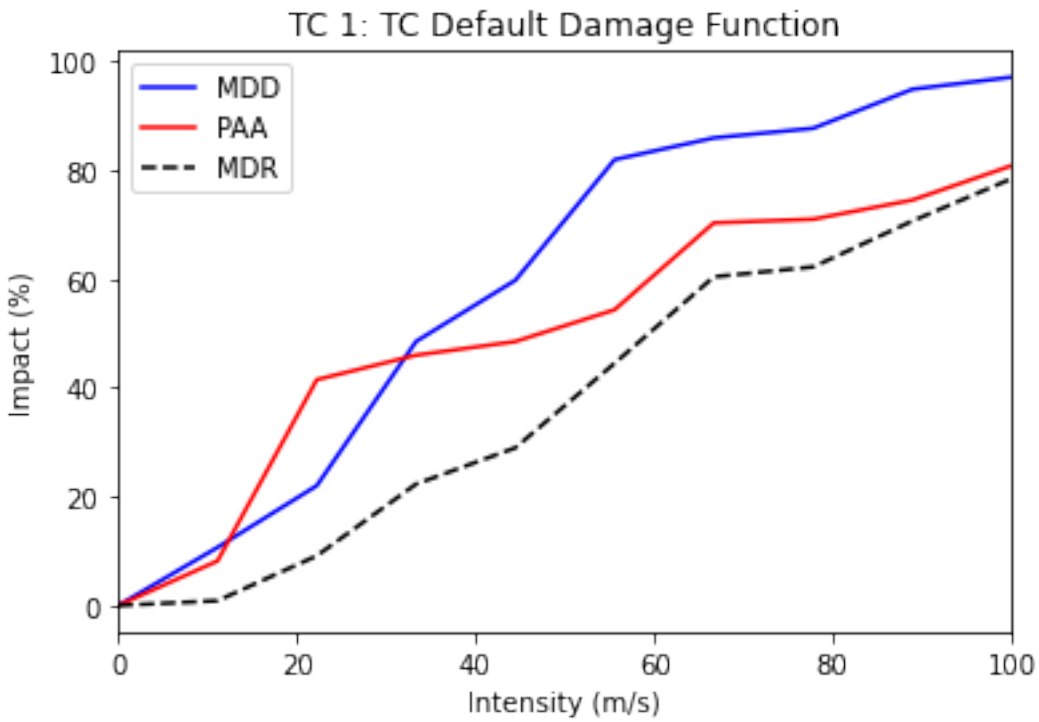
```
# plotting all the impact functions in impf_set
axes = imp_fun_set.plot()
```



Retrieving an impact function from the `ImpactFuncSet`

User may want to retrive a particular impact function from `ImpactFuncSet`. Using the method `get_func(haz_type, id)`, it returns an `ImpactFunc` class of the desired impact function. Below is an example of extracting the TC impact function with id 1, and using `plot()` to visualise the function.

```
# extract the TC impact function with id 1
impf_tc_1 = imp_fun_set.get_func("TC", 1)
# plot the impact function
impf_tc_1.plot();
```

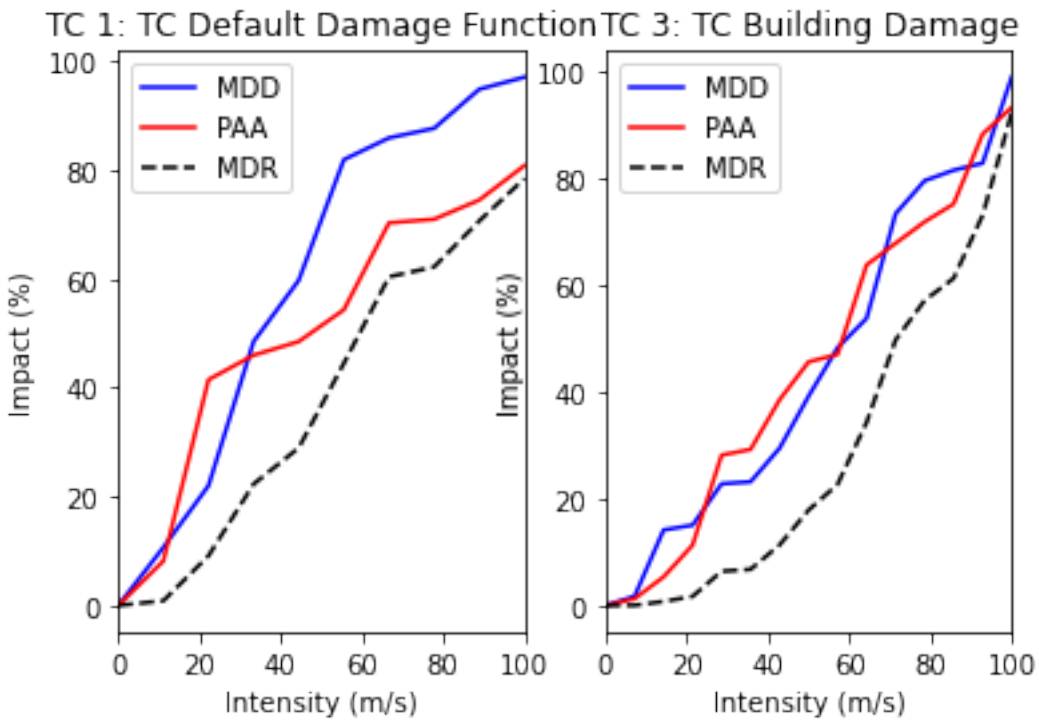


Removing an impact function from the `ImpactFuncSet`

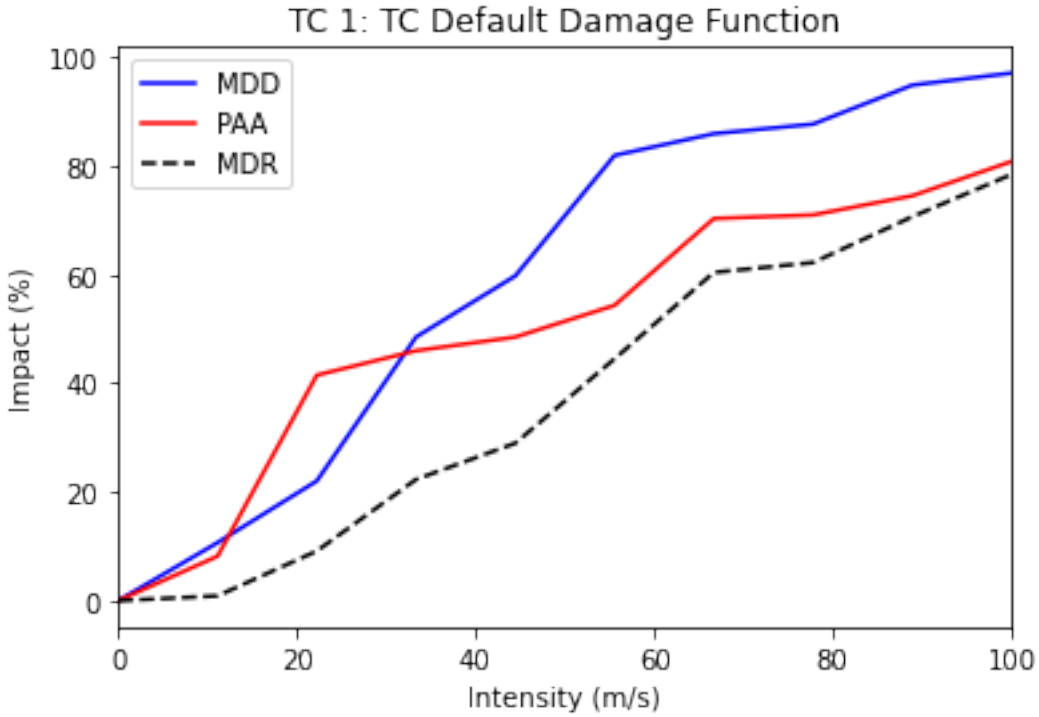
If there is an unwanted impact function from the `ImpactFuncSet`, we may remove it using the method `remove_func(haz_type, id)` to remove it from the set.

For example, in the previous generated impact function set `imp_fun_set` contains an unwanted TC impact function with id 3, we might thus would like to remove that from the set.

```
# first plotting all the impact functions in the impact function set to see what is␣
↪in there:
imp_fun_set.plot();
```




```
# removing the TC impact function with id 3
imp_fun_set.remove_func("TC", 3)
# plot all the remaining impact functions in imp_fun_set
imp_fun_set.plot();
```



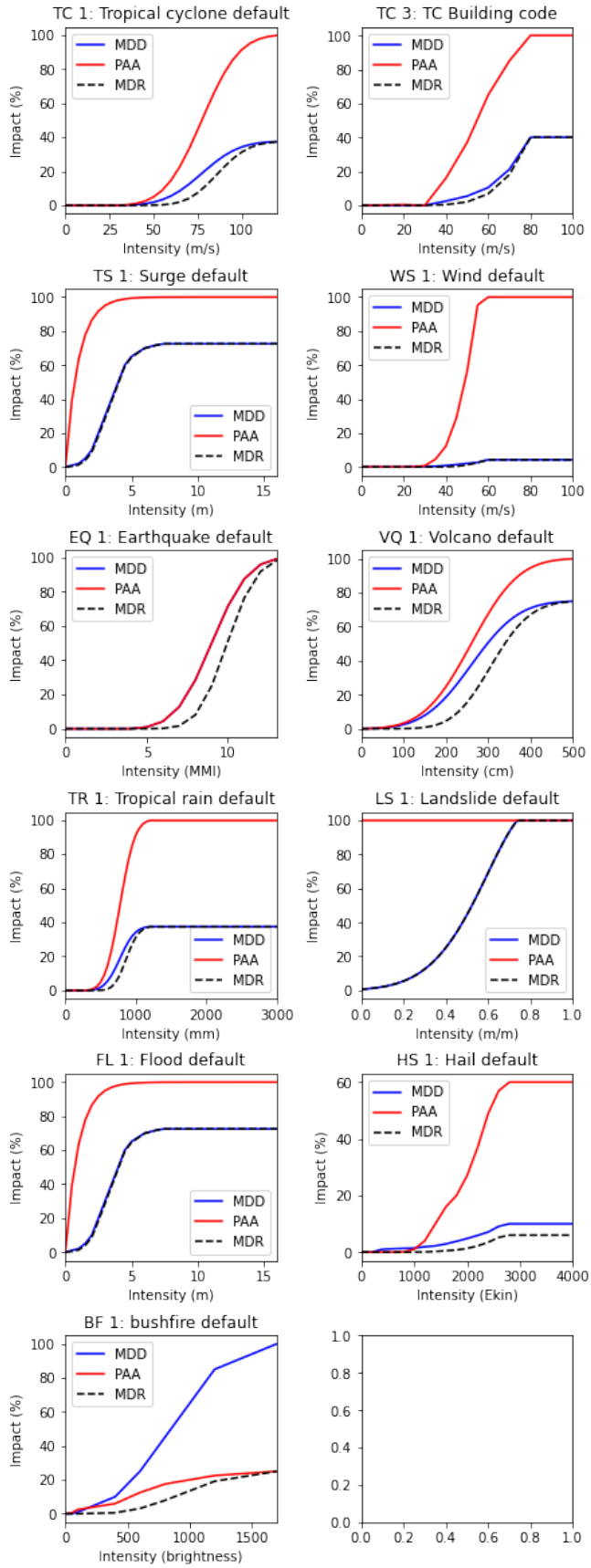
Part 4: Read and write `ImpactFuncSet` into Excel sheets

Users may load impact functions to an `ImpactFuncSet` class from an excel sheet, or to write the `ImpactFuncSet` into an excel sheet. This section will give an example of how to do it.

Reading impact functions from an Excel file

Impact functions defined in an excel file following the template provided in sheet `impact_functions` of `climada_python/climada/data/system/entity_template.xlsx` can be ingested directly using the method `from_excel()`.

```
from climada.entity import ImpactFuncSet
from climada.util import ENT_TEMPLATE_XLS
import matplotlib.pyplot as plt

# provide absolute path of the input excel file
file_name = ENT_TEMPLATE_XLS
# fill ImpactFuncSet from Excel file
imp_set_xlsx = ImpactFuncSet.from_excel(file_name)

# plot all the impact functions from the ImpactFuncSet
imp_set_xlsx.plot()
# adjust the plots
plt.subplots_adjust(right=1.0, top=4.0, hspace=0.4, wspace=0.4)
```


TC 1: Tropical cyclone default
Impact (%)
Intensity (m/s)
MDD
PAA
MDR
TC 3: TC Building code
Impact (%)
Intensity (m/s)
MDD
PAA
MDR
TS 1: Surge default
Impact (%)
Intensity (m)
MDD
PAA
MDR
WS 1: Wind default
Impact (%)
Intensity (m/s)
MDD
PAA
MDR
EQ 1: Earthquake default
Impact (%)
Intensity (MMI)
MDD
PAA
MDR
VQ 1: Volcano default
Impact (%)
Intensity (cm)
MDD
PAA
MDR
TR 1: Tropical rain default
Impact (%)
Intensity (mm)
MDD
PAA
MDR
LS 1: Landslide default
Impact (%)
Intensity (m/m)
MDD
PAA
MDR
FL 1: Flood default
Impact (%)
Intensity (m)
MDD
PAA
MDR
HS 1: Hail default
Impact (%)
Intensity (Ekin)
MDD
PAA
MDR
BF 1: bushfire default
Impact (%)
Intensity (brightness)
MDD
PAA
MDR

Write impact functions

Users may write the impact functions in Excel format using `write_excel()` method.

```
# write imp_set_xlsx into an excel file
imp_set_xlsx.write_excel("tutorial_impf_set.xlsx")
```

Alternative saving format

Alternatively, users may also save the impact functions into pickle format, using CLIMADA in-built function `save()`. Note that pickle has a *transient format* and should be avoided when possible.

```
from climada.util.save import save

# this generates a results folder in the current path and stores the output there
save("tutorial_impf_set.p", imp_set_xlsx)
```

```
2022-03-28 20:08:56,846 - climada.util.save - INFO - Written file /mnt/c/users/yyljy/
↪documents/climada_main/doc/tutorial/results/tutorial_impf_set.p
```

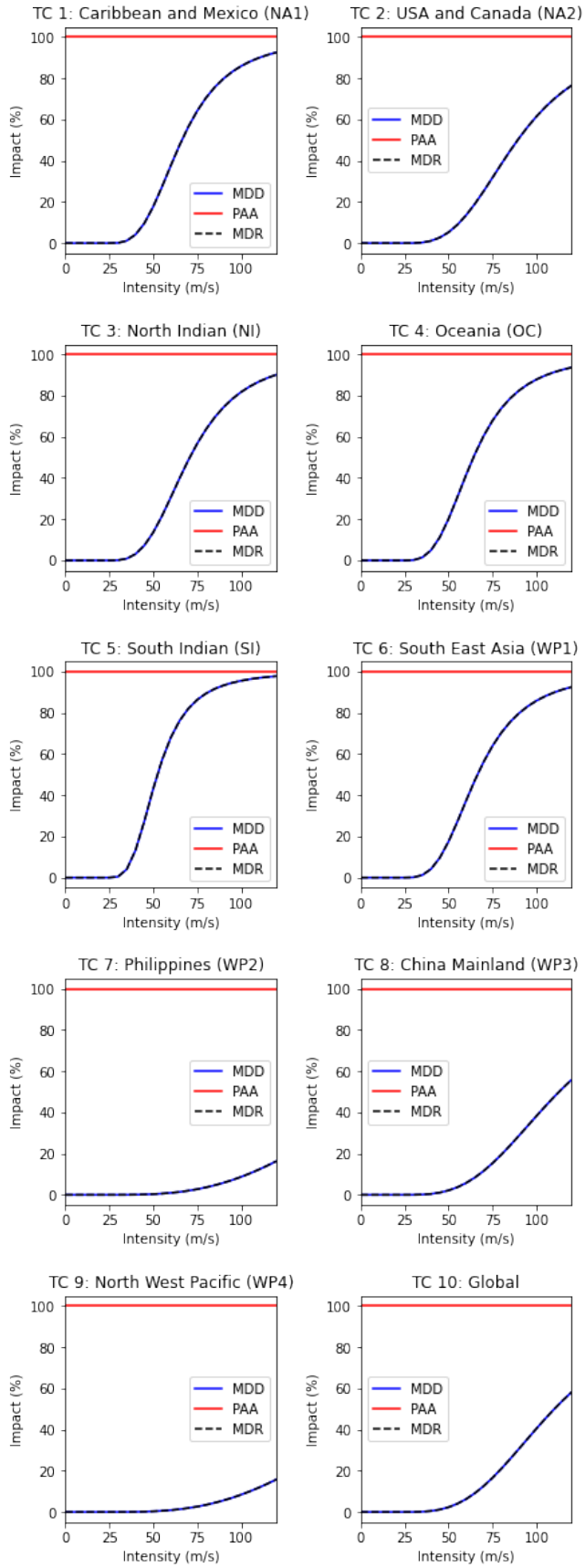
Part 5: Loading `ImpactFuncSet` from CLIMADA in-built impact functions

Similar to *Part 3*, some of the impact functions are available as `ImpactFuncSet` classes. Users may load them from the CLIMADA modules.

Here we use the example of the calibrated impact functions of TC wind damages per region to property damages, following the reference Eberenz et al. (2021). Method `from_calibrated_regional_ImpfSet()` returns a set of default calibrated impact functions for TC for different regions.

```
from climada.entity.impact_funcs.trop_cyclone import ImpfSetTropCyclone
import matplotlib.pyplot as plt

# generate the default calibrated TC impact functions for different regions
imp_fun_set_TC = ImpfSetTropCyclone.from_calibrated_regional_ImpfSet()

# plot all the impact functions
imp_fun_set_TC.plot()
# adjust the plots
plt.subplots_adjust(right=1.0, top=4.0, hspace=0.4, wspace=0.4)
```

```
/tmp/ipykernel_1009/2983082256.py:10: UserWarning: Tight layout not applied. tight_
↪layout cannot make axes height small enough to accommodate all axes decorations.
  plt.tight_layout()
```


TC 1: Caribbean and Mexico (NA1)
TC 2: USA and Canada (NA2)
TC 3: North Indian (NI)
TC 4: Oceania (OC)
TC 5: South Indian (SI)
TC 6: South East Asia (WP1)
TC 7: Philippines (WP2)
TC 8: China Mainland (WP3)
TC 9: North West Pacific (WP4)
TC 10: Global
Impact (%)
Intensity (m/s)
MDD
PAA
MDR

2.5.3 Adaptation Measures

Adaptation measures are defined by parameters that alter the exposures, hazard or impact functions. Risk transfer options are also considered. Single measures are defined in the `Measure` class, which can be aggregated to a `MeasureSet`.

Measure class

A measure is characterized by the following attributes:

Related to measure's description:

- name (str): name of the action
- haz_type (str): related hazard type (peril), e.g. TC
- color_rgb (np.array): integer array of size 3. Gives color code of this measure in RGB
- cost (float): discounted cost (in same units as assets). Needs to be provided by the user. See the example provided in `climada_python/climada/data/system/entity_template.xlsx` sheets `_measures_details` and `_discounting_sheet` to see how the discounting is done.

Related to a measure's impact:

- hazard_set (str): file name of hazard to use
- hazard_freq_cutoff (float): hazard frequency cutoff
- exposure_set (str): file name of exposure to use
- hazard_inten_imp (tuple): parameter a and b of hazard intensity change
- mdd_impact (tuple): parameter a and b of the impact over the mean damage degree
- paa_impact (tuple): parameter a and b of the impact over the percentage of affected assets
- imp_fun_map (str): change of impact function id, e.g. '1to3'
- exp_region_id (int): region id of the selected exposures to consider ALL the previous parameters
- risk_transf_attach (float): risk transfer attachment. Applies to the whole exposure.
- risk_transf_cover (float): risk transfer cover. Applies to the whole exposure.

Parameters description:

`hazard_set` and `exposures_set` provide the file names in h5 format (generated by CLIMADA) of the hazard and exposures to use as a result of the implementation of the measure. These might be further modified when applying the other parameters.

`hazard_inten_imp`, `mdd_impact` and `paa_impact` transform the impact functions linearly as follows:

```
intensity = intensity*hazard_inten_imp[0] + hazard_inten_imp[1]
mdd = mdd*mdd_impact[0] + mdd_impact[1]
paa = paa*paa_impact[0] + paa_impact[1]
```

`hazard_freq_cutoff` modifies the hazard by putting 0 intensities to the events whose impact exceedance frequency are greater than `hazard_freq_cutoff`.

`imp_fun_map` indicates the ids of the impact function to replace and its replacement. The `impf_XX` variable of `Exposures` with the affected impact function id will be correspondingly modified (`XX` refers to the `haz_type` of the measure).

`exp_region_id` will apply all the previous changes only to the `region_id` indicated. This means that only the exposures with that `region_id` and the hazard's centroids close to them will be modified with the previous changes, the other regions will remain unaffected to the measure.

`risk_transf_attach` and `risk_transf_cover` are the deductible and coverage of any event to happen.

Methods description:

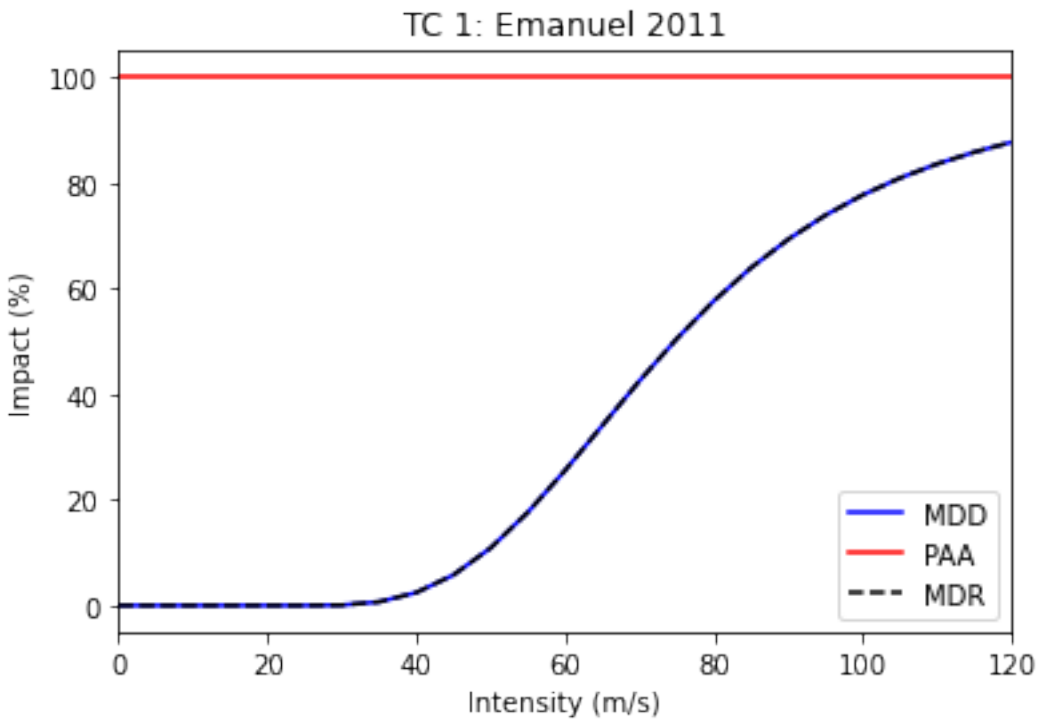
The method `check()` validates the attibutes. `apply()` applies the measure to a given exposure, impact function and hazard, returning their modified values. The parameters related to insurability (risk_transf_attach and risk_transf_cover) affect the resulting impact and are therefore not applied in the `apply()` method yet.

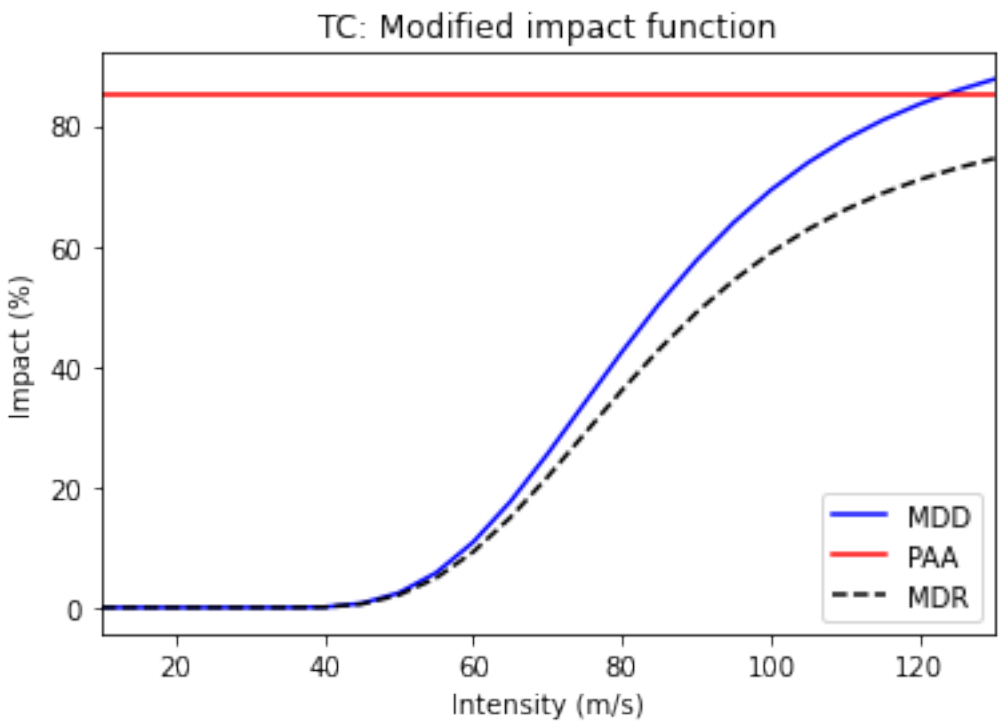
`calc_impact()` calls to `apply()`, applies the insurance parameters and returns the final impact and risk transfer of the measure. This method is called from the `CostBenefit` class.

The method `apply()` allows to visualize the effect of a measure. Here are some examples:

```
# effect of mdd_impact, paa_impact, hazard_inten_imp
%matplotlib inline
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard

# define measure
meas = Measure(
    name="Mangrove",
    haz_type="TC",
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    mdd_impact=(1, 0),
    paa_impact=(1, -0.15),
    hazard_inten_imp=(1, -10),  # reduces intensity by 10
)

# impact functions
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_all = ImpactFuncSet([impf_tc])
impf_all.plot()

# dummy Hazard and Exposures
haz = Hazard("TC")  # this measure does not change hazard
exp = Exposures()  # this measure does not change exposures

# new impact functions
new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)
axes = new_impfs.plot()
axes.set_title("TC: Modified impact function");
```

```
Text(0.5, 1.0, 'TC: Modified impact function')
```






```
# effect of hazard_freq_cutoff
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
```

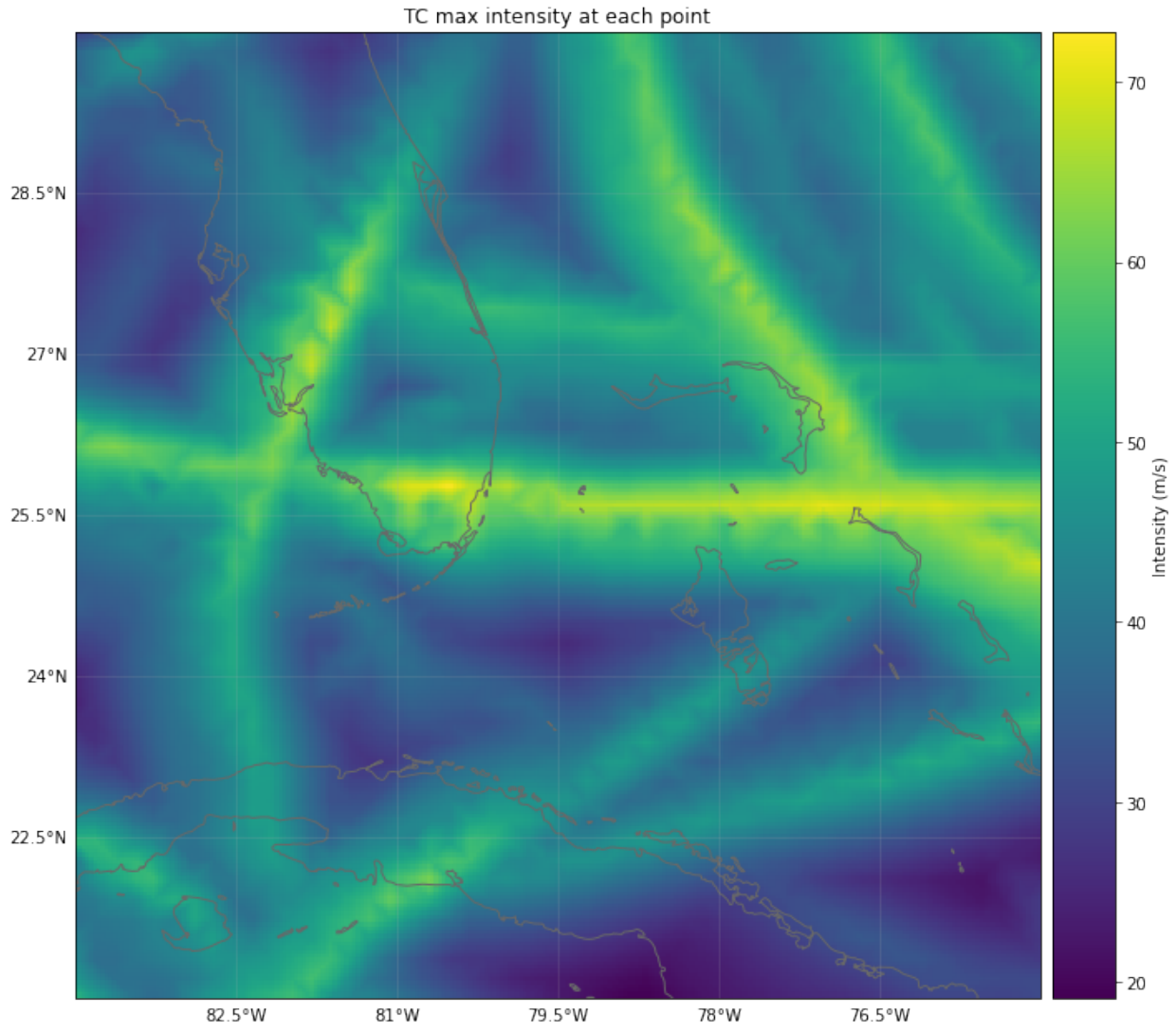
(continues on next page)

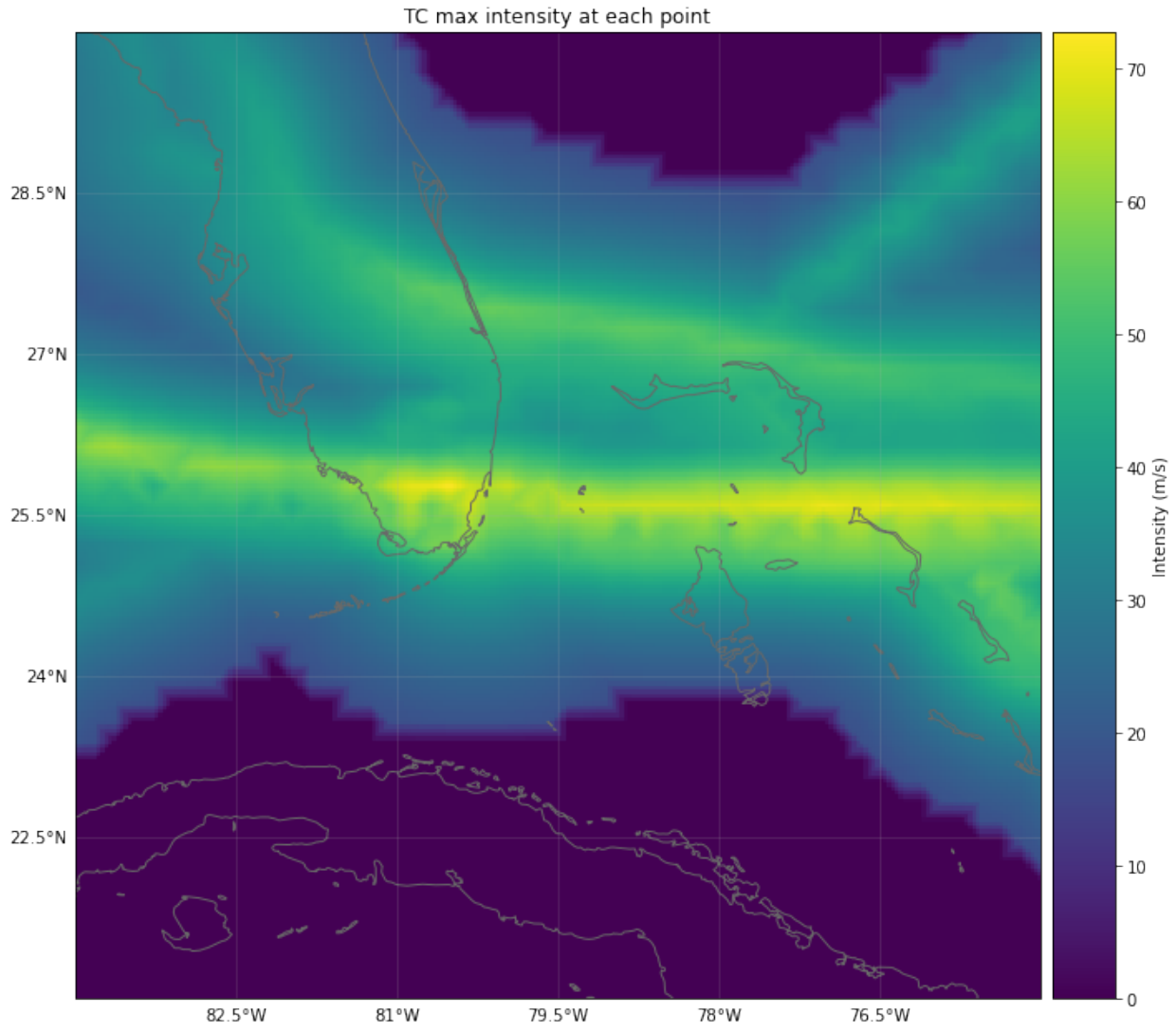
(continued from previous page)

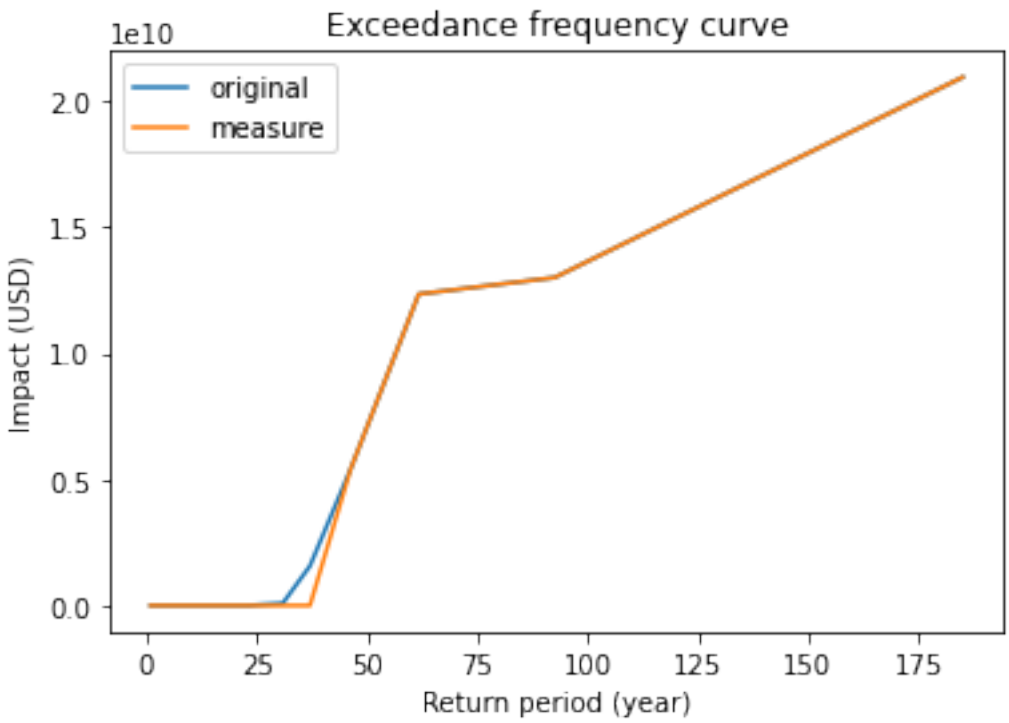
```
from climada.engine import ImpactCalc

from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure(
    name="Mangrove",
    haz_type="TC",
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    hazard_freq_cutoff=0.0255,
)

# impact functions
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_all = ImpactFuncSet([impf_tc])

# Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures.from_hdf5(EXP_DEMO_H5)
exp.check()

# new hazard
new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)
# if you look at the maximum intensity per centroid: new_haz does not contain the␣
↪event with smaller impact (the most frequent)
haz.plot_intensity(0)
new_haz.plot_intensity(0)
# you might also compute the exceedance frequency curve of both hazard
imp = ImpactCalc(exp, impf_all, haz).impact()
ax = imp.calc_freq_curve().plot(label="original")

new_imp = ImpactCalc(new_exp, new_impfs, new_haz).impact()
new_imp.calc_freq_curve().plot(
    axis=ax, label="measure"
);  # the damages for events with return periods > 1/0.0255 ~ 40 are 0
```

```
$CONDA_PREFIX/lib/python3.8/site-packages/pyproj/crs/crs.py:68: FutureWarning: '+init=
↪<authority>:<code>' syntax is deprecated. '<authority>:<code>' is the preferred␣
↪initialization method. When making the change, be mindful of axis order changes:␣
↪https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-in-proj-6
  return _prepare_from_string(" ".join(pjargs))
```

```
<matplotlib.legend.Legend at 0x7f433756d970>
```


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
Intensity (m/s)


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
10
0
Intensity (m/s)




```
# effect of exp_region_id
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
from climada.engine import ImpactCalc

from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure(
    name="Building code",
    haz_type="TC",
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    hazard_freq_cutoff=0.00455,
    exp_region_id=[1],  # apply measure to points close to exposures with region_id=1
)

# impact functions
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_all = ImpactFuncSet([impf_tc])

# Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures.from_hdf5(EXP_DEMO_H5)
```

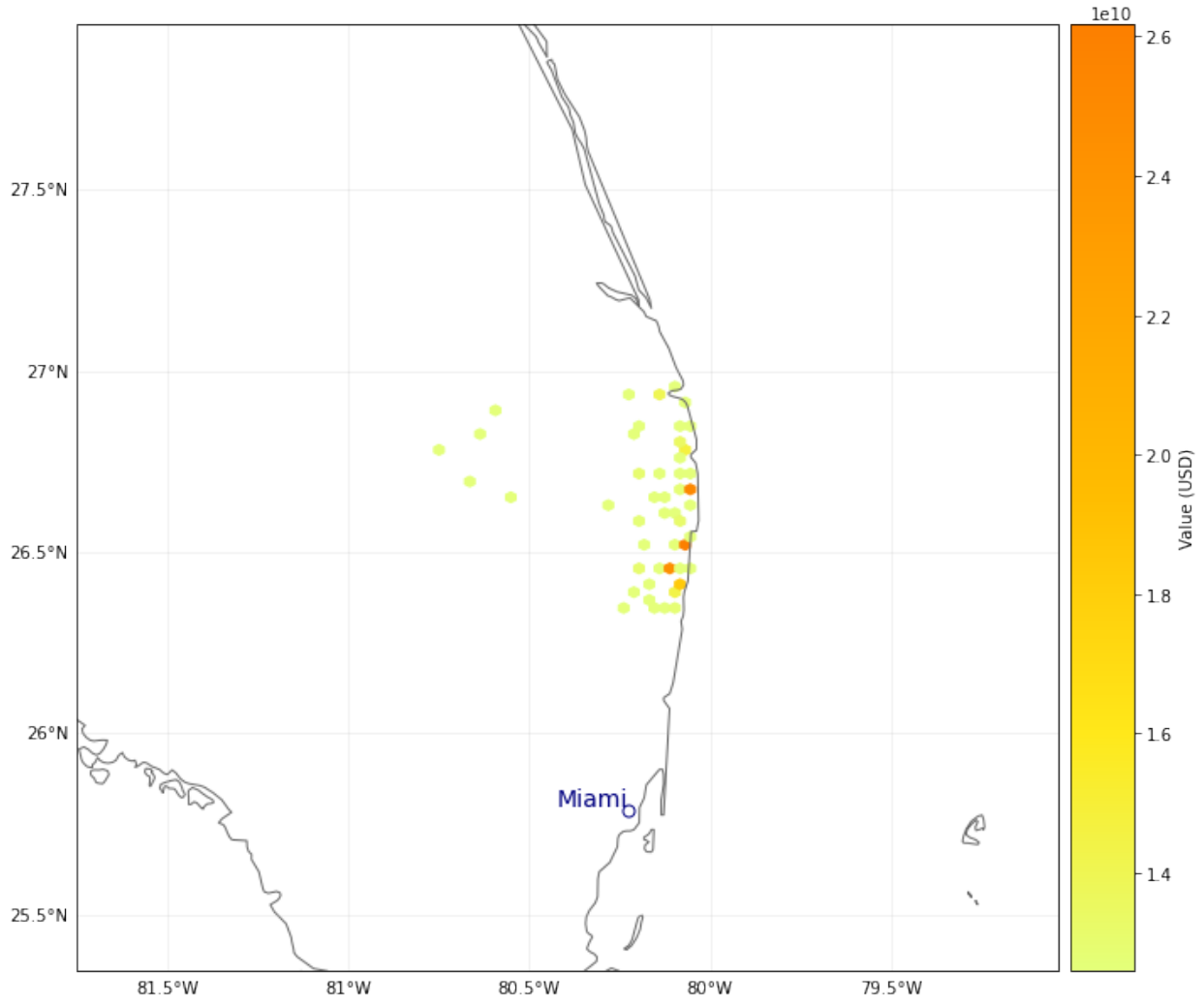
(continues on next page)

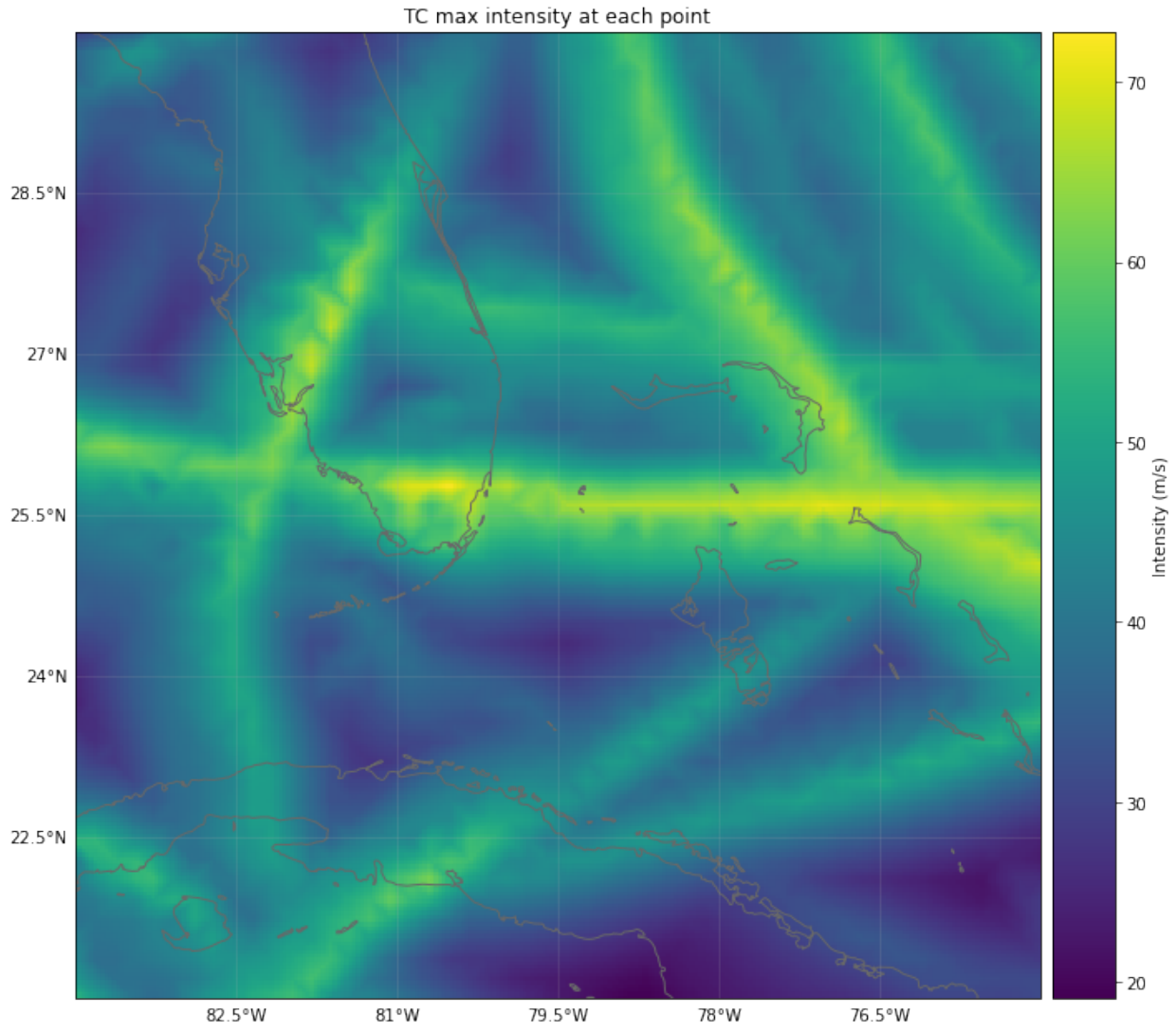
(continued from previous page)

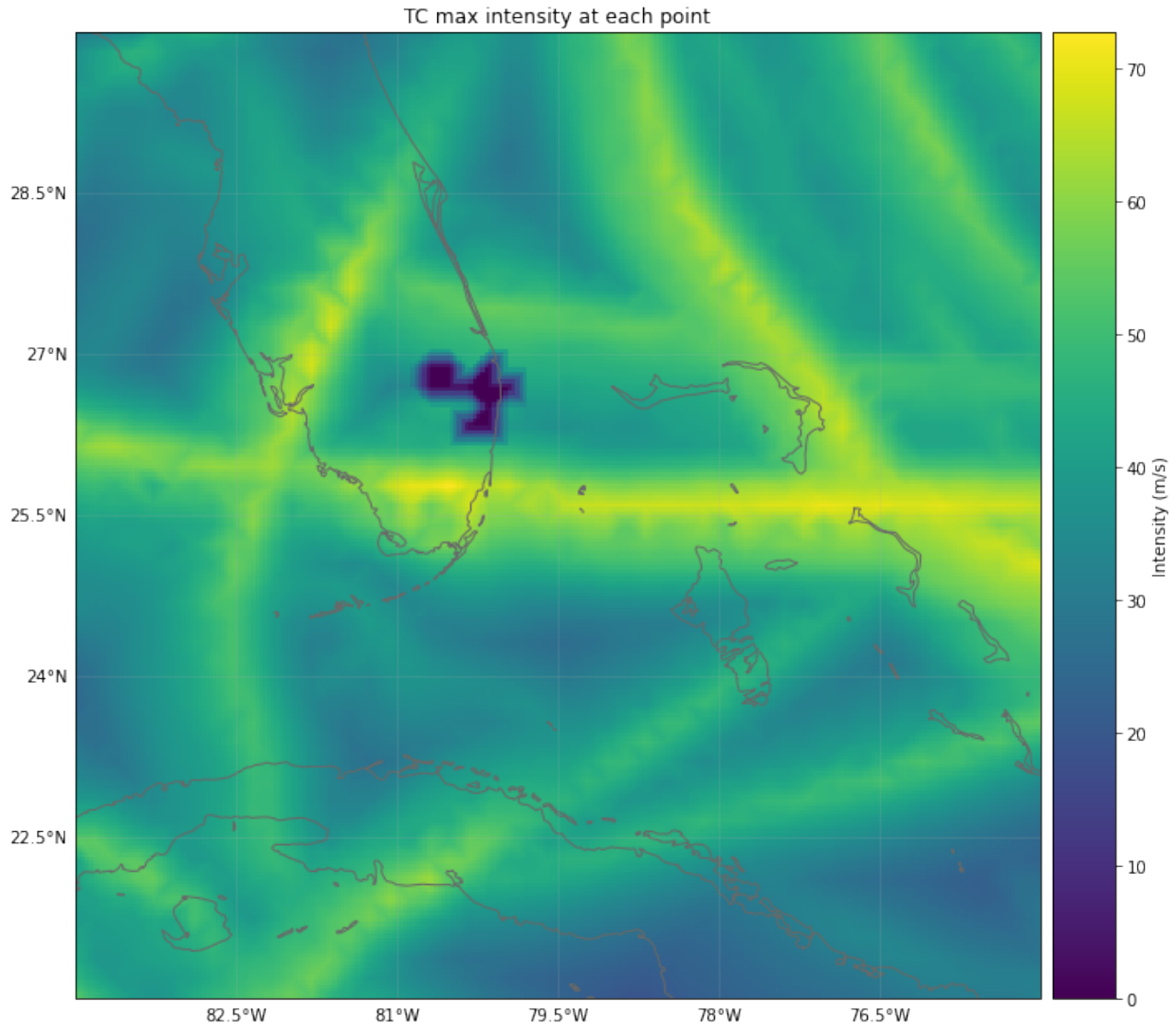
```
# exp['region_id'] = np.ones(exp.shape[0])
exp.check()
# all exposures have region_id=1
exp.plot_hexbin(buffer=1.0)

# new hazard
new_exp, new_impfs, new_haz = meas.apply(exp, impf_all, haz)
# the cutoff has been apllied only in the region of the exposures
haz.plot_intensity(0)
new_haz.plot_intensity(0)

# the exceddance frequency has only been computed for the selected exposures before␣
↪doing the cutoff.
# since we have removed the hazard of the places with exposure, the new exceedance␣
↪frequency curve is zero.
imp = ImpactCalc(exp, impf_all, haz).impact()
imp.calc_freq_curve().plot()

new_imp = ImpactCalc(new_exp, new_impfs, new_haz).impact()
new_imp.calc_freq_curve().plot();
```

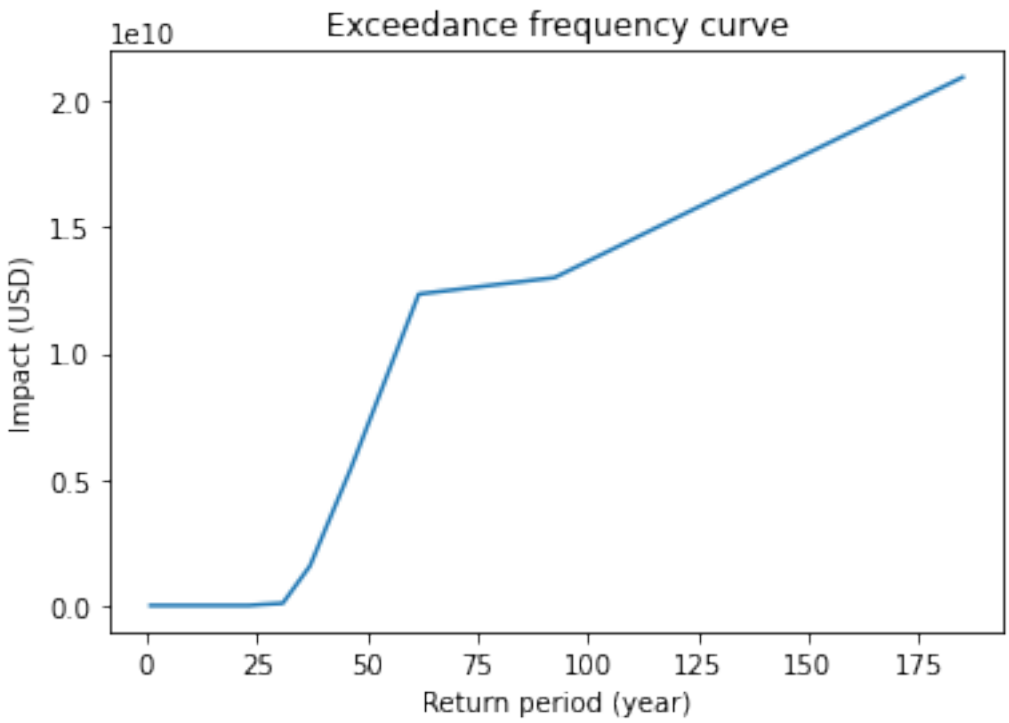
```
$CONDA_PREFIX/lib/python3.8/site-packages/pyproj/crs/crs.py:68: FutureWarning: '+init=
↪<authority>:<code>' syntax is deprecated. '<authority>:<code>' is the preferred␣
↪initialization method. When making the change, be mindful of axis order changes:␣
↪https://pyproj4.github.io/pyproj/stable/gotchas.html#axis-order-changes-in-proj-6
  return _prepare_from_string(" ".join(pjargs))
```

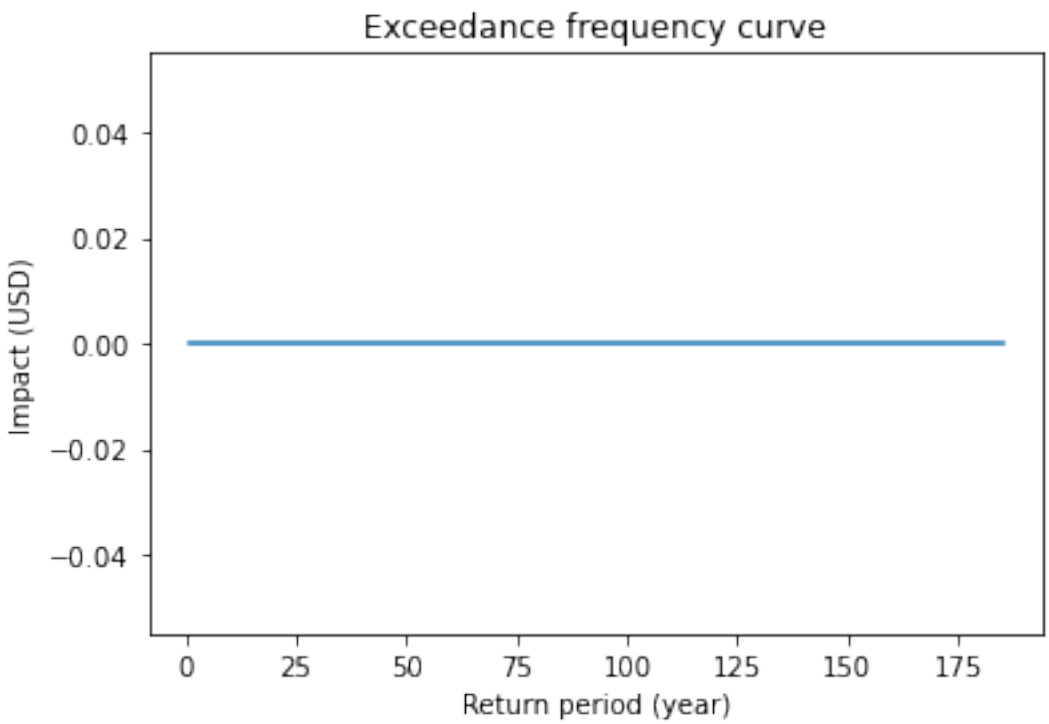
```
<AxesSubplot:title={'center':'Exceedance frequency curve'}, xlabel='Return period␣
↪(year)', ylabel='Impact (USD)'>
```


1e10
2.6
2.4
2.2
2.0
1.8
1.6
1.4
Value (USD)
27.5°N
27°N
26.5°N
26°N
25.5°N
81.5°W
81°W
80.5°W
80°W
79.5°W
Miami


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
Intensity (m/s)


TC max intensity at each point
28.5°N
27°N
25.5°N
24°N
22.5°N
82.5°W
81°W
79.5°W
78°W
76.5°W
70
60
50
40
30
20
10
0
Intensity (m/s)





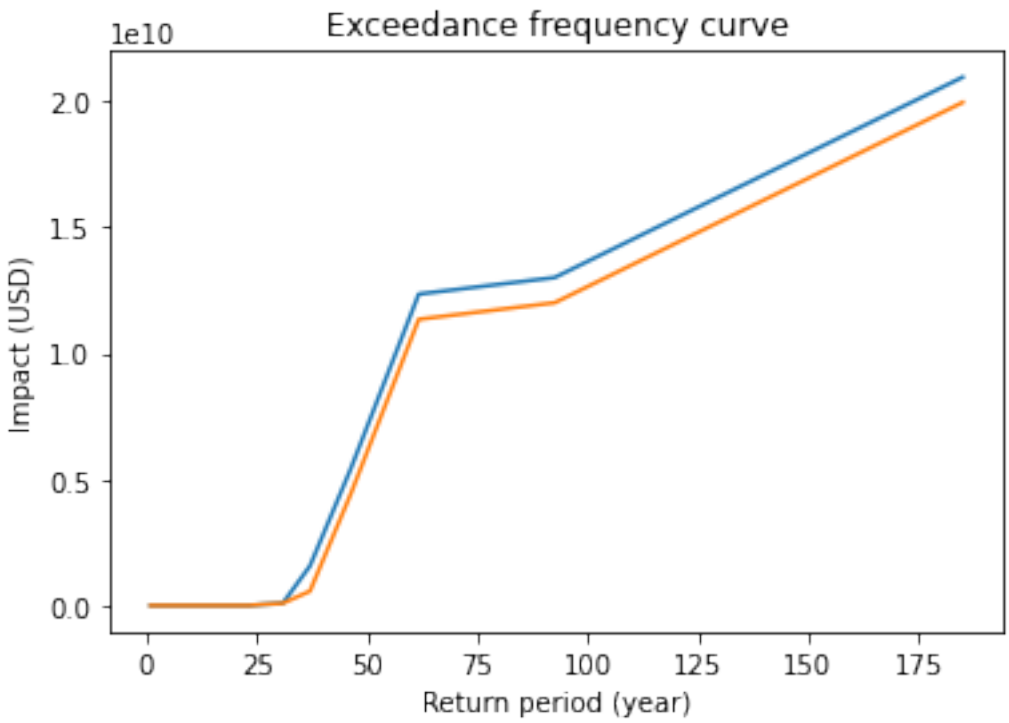

```
# effect of risk_transf_attach and risk_transf_cover
import numpy as np
from climada.entity import ImpactFuncSet, ImpfTropCyclone, Exposures
from climada.entity.measures import Measure
from climada.hazard import Hazard
```

(continues on next page)

(continued from previous page)

```
from climada.engine import ImpactCalc

from climada.util import HAZ_DEMO_H5, EXP_DEMO_H5

# define measure
meas = Measure(
    name="Insurance",
    haz_type="TC",
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    risk_transf_attach=5.0e8,
    risk_transf_cover=1.0e9,
)

# impact functions
impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_all = ImpactFuncSet([impf_tc])

# Hazard
haz = Hazard.from_hdf5(HAZ_DEMO_H5)
haz.check()

# Exposures
exp = Exposures.from_hdf5(EXP_DEMO_H5)
exp.check()

# impact before
imp = ImpactCalc(exp, impf_all, haz).impact()
ax = imp.calc_freq_curve().plot(label="original")

# impact after. risk_transf will be added to the cost of the measure
imp_new, risk_transf = meas.calc_impact(exp, impf_all, haz)
imp_new.calc_freq_curve().plot(axis=ax, label="measure")
print("risk_transfer {:.3}".format(risk_transf.aai_agg))
```

```
2022-03-30 20:10:29,899 - climada.hazard.base - INFO - Reading /home/yuyue/climada/
↪demo/data/tc_fl_1990_2004.h5
2022-03-30 20:10:30,001 - climada.entity.exposures.base - INFO - Reading /home/yuyue/
↪climada/demo/data/exp_demo_today.h5
2022-03-30 20:10:30,030 - climada.entity.exposures.base - INFO - centr_ not set.
2022-03-30 20:10:30,034 - climada.entity.exposures.base - INFO - Matching 50␣
↪exposures with 2500 centroids.
2022-03-30 20:10:30,035 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-03-30 20:10:30,047 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 216 events.
2022-03-30 20:10:30,084 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-30 20:10:30,087 - climada.engine.impact - INFO - Calculating damage for 50␣
↪assets (>0) and 216 events.
risk_transfer 2.7e+07
```



MeasureSet class

Similarly to the `ImpactFuncSet`, `MeasureSet` is a container which handles `Measure` instances through the methods `append()`, `extend()`, `remove_measure()` and `get_measure()`. Use the `check()` method to make sure all the measures have been properly set.

For a complete class documentation, refer to the Python modules docs: *climada.entity.measures.measure_set.MeasureSet*

```
# build measures
import numpy as np
import matplotlib.pyplot as plt
from climada.entity.measures import Measure, MeasureSet

meas_1 = Measure(
    haz_type="TC",
    name="Mangrove",
    color_rgb=np.array([1, 1, 1]),
    cost=500000000,
    mdd_impact=(1, 2),
    paa_impact=(1, 2),
    hazard_inten_imp=(1, 2),
    risk_transf_cover=500,
)

meas_2 = Measure(
    haz_type="TC",
    name="Sandbags",
    color_rgb=np.array([1, 1, 1]),
    cost=22000000,
```

(continues on next page)

(continued from previous page)

```
    mdd_impact=(1, 2),
    paa_impact=(1, 3),
    hazard_inten_imp=(1, 2),
    exp_region_id=2,
)

# gather all measures
meas_set = MeasureSet()
meas_set.append(meas_1)
meas_set.append(meas_2)
meas_set.check()

# select one measure
meas_sel = meas_set.get_measure(name="Sandbags")
print(meas_sel[0].name, meas_sel[0].cost)
```

```
Sandbags 22000000
```

Read/write measure sets to/from Excel files

Measures defined in an excel file following the template provided in sheet `measures` of `climada_python/data/system/entity_template.xlsx` can be ingested directly using the method `from_excel()`. Measure sets can be written to file with the method `write_excel()`.

```
from climada.entity.measures import MeasureSet
from climada.util import ENT_TEMPLATE_XLS

# Fill DataFrame from Excel file
file_name = ENT_TEMPLATE_XLS  # provide absolute path of the excel file
meas_set = MeasureSet.from_excel(file_name)
```

2.5.4 DiscRates class

Discount rates are used to calculate the net present value of any future or past value. They are thus used to compare amounts paid (costs) and received (benefits) in different years. A project is economically viable (attractive), if the net present value of benefits exceeds the net present value of costs - a cost-benefit ratio < 1.

There are several important implications that come along with discount rates. Namely, that higher discount rates lead to smaller net present values of future impacts (costs). As a consequence of that, climate action and mitigation measures can be postboned. In the literature higher discount rates are typically justified by the expectation of continued exponential growth of the economy. The most widley used interest rate in climate change economics is 1.4% as propsed by the Stern Review (2006). Neoliberal economists around Nordhaus (2007) claim that rates should be higher, around 4.3%. Environmental economists argue that future costs shouldn't be discounted at all. This is especially true for non-monetary variables such as ecosystems or human lifes, where no price tag should be applied out of ethical reasons. This discussion has a long history, reaching back to the 18th century: "Some things have a price, or relative worth, while other things have a dignity, or inner worth" (Kant, 1785).

This class contains the discount rates for every year and discounts given values. Its attributes are:

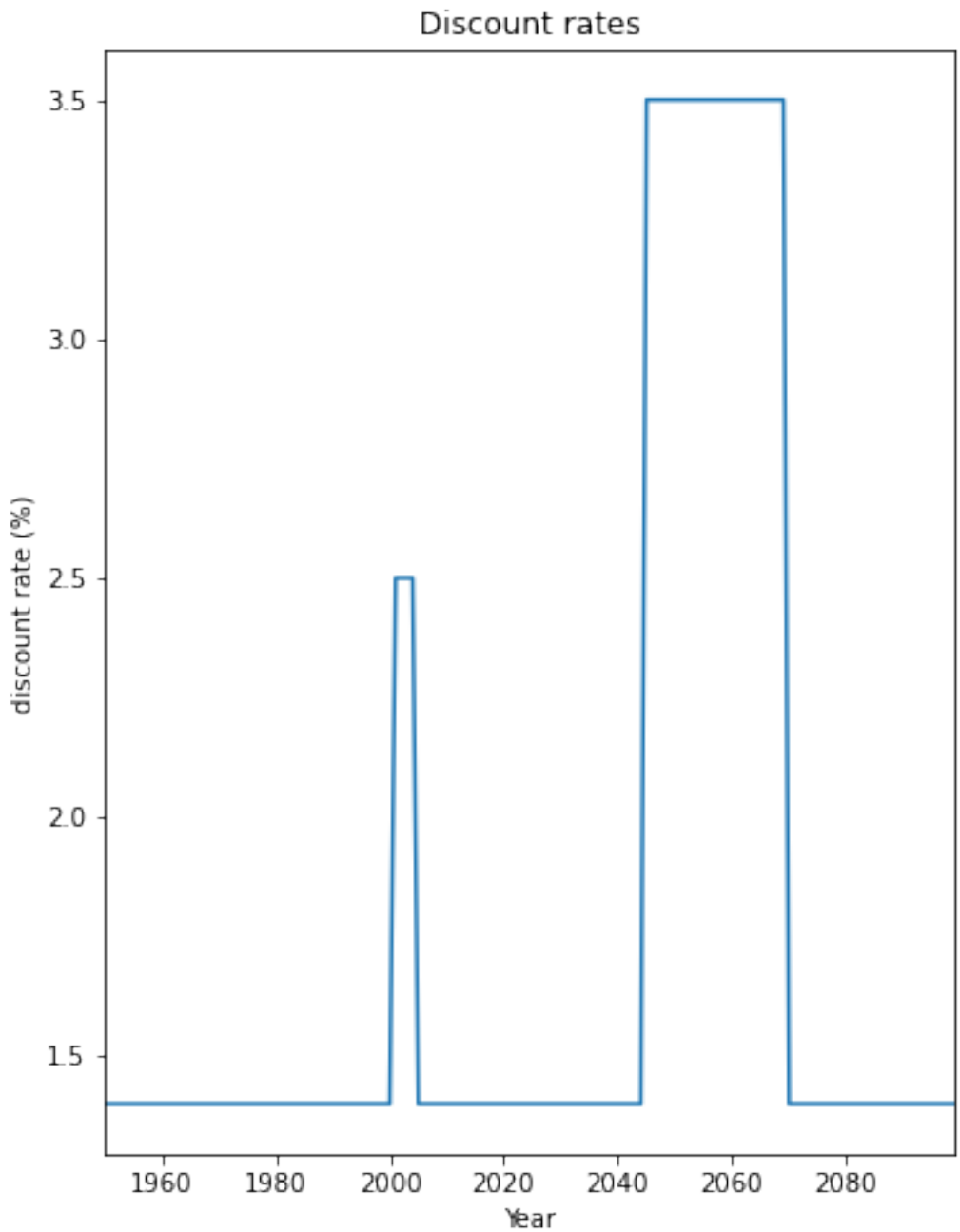
- years (np.array): years
- rates (np.array): discount rates for each year (between 0 and 1)

For a complete class documentation, refer to the Python modules docs: *climada.entity.disc_rates.base.DiscRates*

An example of use - we define discount rates and apply them on a coastal protection scheme which initially costs 100 mn. USD plus 75'000 USD mainteance each year, starting after 10 years. Net present value of the project can be calculated as displayed:

```
%matplotlib inline
import numpy as np
from climada.entity import DiscRates

# define discount rates
years = np.arange(1950, 2100)
rates = np.ones(years.size) * 0.014
rates[51:55] = 0.025
rates[95:120] = 0.035
disc = DiscRates(years=years, rates=rates)
disc.plot()

# Compute net present value between present year and future year.
ini_year = 2019
end_year = 2050
val_years = np.zeros(end_year - ini_year + 1)
val_years[0] = 100000000  # initial investment
val_years[10:] = 75000  # maintenance from 10th year
npv = disc.net_present_value(ini_year, end_year, val_years)
print("net present value: {:.5e}".format(npv))
```

```
net present value: 1.01231e+08
```



Read discount rates of an Excel file

Discount rates defined in an excel file following the template provided in sheet `discount` of `climada_python/climada/data/system/entity_template.xlsx` can be ingested directly using the method `from_excel()`.

```
from climada.entity import DiscRates
from climada.util import ENT_TEMPLATE_XLS

# Fill DataFrame from Excel file
file_name = ENT_TEMPLATE_XLS  # provide absolute path of the excel file
print("Read file:", ENT_TEMPLATE_XLS)
```

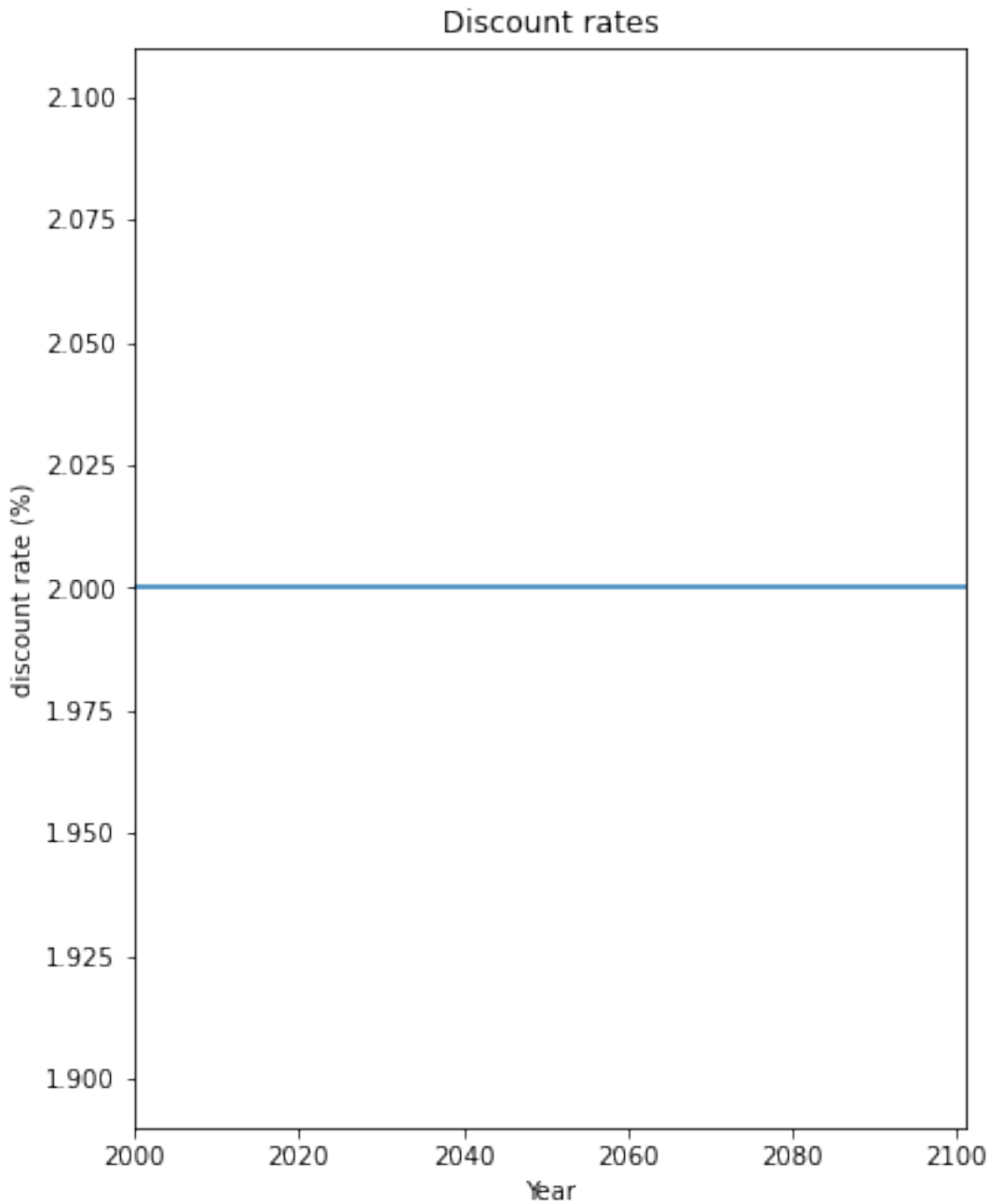
(continues on next page)

(continued from previous page)

```
disc = DiscRates.from_excel(file_name)
disc.plot();
```

```
Read file: /Users/ckropf/climada/data/entity_template.xlsx
```

```
<AxesSubplot:title={'center':'Discount rates'}, xlabel='Year', ylabel='discount rate (
↪%)'>
```



Write discount rates

Users may write the discounts in Excel format using write_excel() method `write_excel()`.

```
from climada.entity import DiscRates
from climada.util import ENT_TEMPLATE_XLS

# Fill DataFrame from Excel file
file_name = ENT_TEMPLATE_XLS  # provide absolute path of the excel file
disc = DiscRates.from_excel(file_name)

# write file
disc.write_excel("results/tutorial_disc.xlsx")
```

Pickle can always be used as well, but note that pickle has a *transient format* and should be avoided when possible:

```
from climada.util.save import save

# this generates a results folder in the current path and stores the output there
save("tutorial_disc.p", disc)
```

2.5.5 Impact Data functionalities

Import data from EM-DAT CSV file and populate Impact()-object with the data.

The core functionality of the module is to read disaster impact data as downloaded from the International Disaster Database EM-DAT (www.emdat.be) and produce a CLIMADA Impact()-instance from it. The purpose is to make impact data easily available for comparison with simulated impact inside CLIMADA, e.g. for calibration purposes.

Data Source

The International Disaster Database EM-DAT www.emdat.be

Download: https://public.emdat.be/ (register for free and download data to continue)

Most important functions

- clean_emdat_df: read CSV from EM-DAT into a DataFrame and clean up.
- emdat_to_impact: create Impact-instance populated with impact data from EM-DAT data (CSV).
- emdat_countries_by_hazard: get list of countries affected by a certain haazrd (disaster (sub-)type) in EM-DAT.
- emdat_impact_yearlysum: create DataFrame with impact from EM-DAT summed per country and year.

Demo data

The demo data used here (demo_emdat_impact_data_2020.csv) contains entries for the disaster subtype "Tropical cyclone" from 2000 to 2020.

```
"""Load required packages and set path to CSV-file from EM-DAT"""

import numpy as np
import pandas as pd
from matplotlib import pyplot as plt

from climada.util.constants import DEMO_DIR
```

(continues on next page)

(continued from previous page)

```
from climada.engine.impact_data import (
    emdat_countries_by_hazard,
    emdat_impact_yearlysum,
    emdat_to_impact,
    clean_emdat_df,
)

# set path to CSV file downloaded from https://public.emdat.be :
emdat_file_path = DEMO_DIR.joinpath("demo_emdat_impact_data_2020.csv")
```

clean_emdat_df()

read CSV from EM-DAT into a DataFrame and clean up.

Use the parameters countries, hazard, and year_range to filter. These parameters are the same for most functions shown here.

```
"""Create DataFrame df with EM-DAT entries of tropical cyclones in Thailand and Viet␣
↪Nam in the years 2005 and 2006"""

df = clean_emdat_df(
    emdat_file_path,
    countries=["THA", "Viet Nam"],
    hazard=["TC"],
    year_range=[2005, 2006],
)
print(df)
```

```
          Dis No  Year  Seq Disaster Group Disaster Subgroup Disaster Type  \
0  2005-0540-VNM  2005  540        Natural    Meteorological         Storm
1  2005-0540-THA  2005  540        Natural    Meteorological         Storm
2  2005-0536-VNM  2005  536        Natural    Meteorological         Storm
3  2005-0611-VNM  2005  611        Natural    Meteorological         Storm
4  2006-0362-VNM  2006  362        Natural    Meteorological         Storm
5  2006-0648-VNM  2006  648        Natural    Meteorological         Storm
6  2006-0251-VNM  2006  251        Natural    Meteorological         Storm
7  2006-0517-VNM  2006  517        Natural    Meteorological         Storm

   Disaster Subtype  Disaster Subsubtype          Event Name Entry Criteria  \
0  Tropical cyclone                  NaN              Damrey           Kill
1  Tropical cyclone                  NaN              Damrey           Kill
2  Tropical cyclone                  NaN             Vicente           Kill
3  Tropical cyclone                  NaN        Kai Tak (21)           Kill
4  Tropical cyclone                  NaN               Bilis           Kill
5  Tropical cyclone                  NaN     Durian (Reming)           Kill
6  Tropical cyclone                  NaN     Chanchu (Caloy)           Kill
7  Tropical cyclone                  NaN  Xangsane (Milenyo)           Kill

   ... End Day Total Deaths No Injured No Affected No Homeless Total Affected  \
0  ...    30.0         75.0       28.0    337632.0         NaN       337660.0
1  ...    30.0         10.0        NaN      2000.0         NaN         2000.0
2  ...    19.0          8.0        NaN      8500.0         NaN         8500.0
```

(continues on next page)

(continued from previous page)

```
3  ...     4.0         20.0        NaN     15000.0         NaN        15000.0
4  ...    19.0         17.0        NaN         NaN      2000.0         2000.0
5  ...     8.0         95.0     1360.0    975000.0    250000.0      1226360.0
6  ...    17.0        204.0        NaN    600000.0         NaN       600000.0
7  ...     6.0         71.0      525.0   1368720.0     98680.0      1467925.0

  Reconstruction Costs ('000 US$) Insured Damages ('000 US$)  \
0                             NaN                        NaN
1                             NaN                        NaN
2                             NaN                        NaN
3                             NaN                        NaN
4                             NaN                        NaN
5                             NaN                        NaN
6                             NaN                        NaN
7                             NaN                        NaN

  Total Damages ('000 US$)        CPI
0                 219250.0  76.388027
1                  20000.0  76.388027
2                  20000.0  76.388027
3                  11000.0  76.388027
4                      NaN  78.852256
5                 456000.0  78.852256
6                      NaN  78.852256
7                 624000.0  78.852256

[8 rows x 43 columns]
```

emdat_countries_by_hazard()

Pick a hazard and a year range to get a list of countries affected from the EM-DAT data.

```
"""emdat_countries_by_hazard: get lists of countries impacted by tropical cyclones␣
↪from 2010 to 2019"""

iso3_codes, country_names = emdat_countries_by_hazard(
    emdat_file_path, hazard="TC", year_range=(2010, 2019)
)

print(country_names)

print(iso3_codes)
```

```
['China', 'Dominican Republic', 'Antigua and Barbuda', 'Fiji', 'Australia',
↪'Bangladesh', 'Belize', 'Barbados', 'Cook Islands', 'Canada', 'Bahamas', 'Guatemala
↪', 'Jamaica', 'Saint Lucia', 'Madagascar', 'Mexico', "Korea, Democratic People's␣
↪Republic of", 'El Salvador', 'Myanmar', 'French Polynesia', 'Solomon Islands',
↪'Taiwan, Province of China', 'India', 'United States of America', 'Honduras', 'Haiti
↪', 'Pakistan', 'Philippines', 'Hong Kong', 'Korea, Republic of', 'Nicaragua', 'Oman
↪', 'Japan', 'Puerto Rico', 'Thailand', 'Martinique', 'Papua New Guinea', 'Tonga',
↪'Venezuela, Bolivarian Republic of', 'Viet Nam', 'Saint Vincent and the Grenadines',
```

(continues on next page)

(continued from previous page)

```
↪ 'Vanuatu', 'Dominica', 'Cuba', 'Comoros', 'Mozambique', 'Malawi', 'Samoa', 'South␣
↪Africa', 'Sri Lanka', 'Palau', 'Wallis and Futuna', 'Somalia', 'Seychelles',
↪'Réunion', 'Kiribati', 'Cabo Verde', 'Micronesia, Federated States of', 'Panama',
↪'Costa Rica', 'Yemen', 'Tuvalu', 'Northern Mariana Islands', 'Colombia', 'Anguilla',
↪ 'Djibouti', 'Cambodia', 'Macao', 'Indonesia', 'Guadeloupe', 'Turks and Caicos␣
↪Islands', 'Saint Kitts and Nevis', "Lao People's Democratic Republic", 'Mauritius',
↪'Marshall Islands', 'Portugal', 'Virgin Islands, U.S.', 'Zimbabwe', 'Saint␣
↪Barthélemy', 'Virgin Islands, British', 'Saint Martin (French part)', 'Sint Maarten␣
↪(Dutch part)', 'Tanzania, United Republic of']
['CHN', 'DOM', 'ATG', 'FJI', 'AUS', 'BGD', 'BLZ', 'BRB', 'COK', 'CAN', 'BHS', 'GTM',
↪'JAM', 'LCA', 'MDG', 'MEX', 'PRK', 'SLV', 'MMR', 'PYF', 'SLB', 'TWN', 'IND', 'USA',
↪'HND', 'HTI', 'PAK', 'PHL', 'HKG', 'KOR', 'NIC', 'OMN', 'JPN', 'PRI', 'THA', 'MTQ',
↪'PNG', 'TON', 'VEN', 'VNM', 'VCT', 'VUT', 'DMA', 'CUB', 'COM', 'MOZ', 'MWI', 'WSM',
↪'ZAF', 'LKA', 'PLW', 'WLF', 'SOM', 'SYC', 'REU', 'KIR', 'CPV', 'FSM', 'PAN', 'CRI',
↪'YEM', 'TUV', 'MNP', 'COL', 'AIA', 'DJI', 'KHM', 'MAC', 'IDN', 'GLP', 'TCA', 'KNA',
↪'LAO', 'MUS', 'MHL', 'PRT', 'VIR', 'ZWE', 'BLM', 'VGB', 'MAF', 'SXM', 'TZA']
```

emdat_to_impact()

function to load EM-DAT impact data and return impact set with impact per event

Parameters:

- emdat_file_csv (str): Full path to EMDAT-file (CSV)
- hazard_type_climada (str): Hazard type abbreviation used in CLIMADA, e.g. 'TC'

Optional parameters:

- hazard_type_emdat (list or str): List of Disaster (sub-)type according EMDAT terminology or CLIMADA hazard type abbreviations. e.g. ['Wildfire', 'Forest fire'] or ['BF']
- year_range (list with 2 integers): start and end year e.g. [1980, 2017]
- countries (list of str): country ISO3-codes or names, e.g. ['JAM', 'CUB']. Set to None or ['all'] for all countries
- reference_year (int): reference year of exposures for normalization. Impact is scaled proportional to GDP to the value of the reference year. No scaling for reference_year=0 (default)
- imp_str (str): Column name of impact metric in EMDAT CSV, e.g. 'Total Affected'; default = "Total Damages"

Returns:

- impact_instance (instance of climada.engine.Impact): Impact() instance (same format as output from CLIMADA impact computations). Values are scaled with GDP to reference_year if reference_year not equal 0. impact_instance.eai_exp holds expected annual impact for each country. impact_instance.coord_exp holds rough central coordinates for each country.
- countries (list): ISO3-codes of countries imn same order as in impact_instance.eai_exp

```
"""Global TC damages 2000 to 2009"""

impact_emdat, countries = emdat_to_impact(
    emdat_file_path, "TC", year_range=(2000, 2009)
```

(continues on next page)

(continued from previous page)

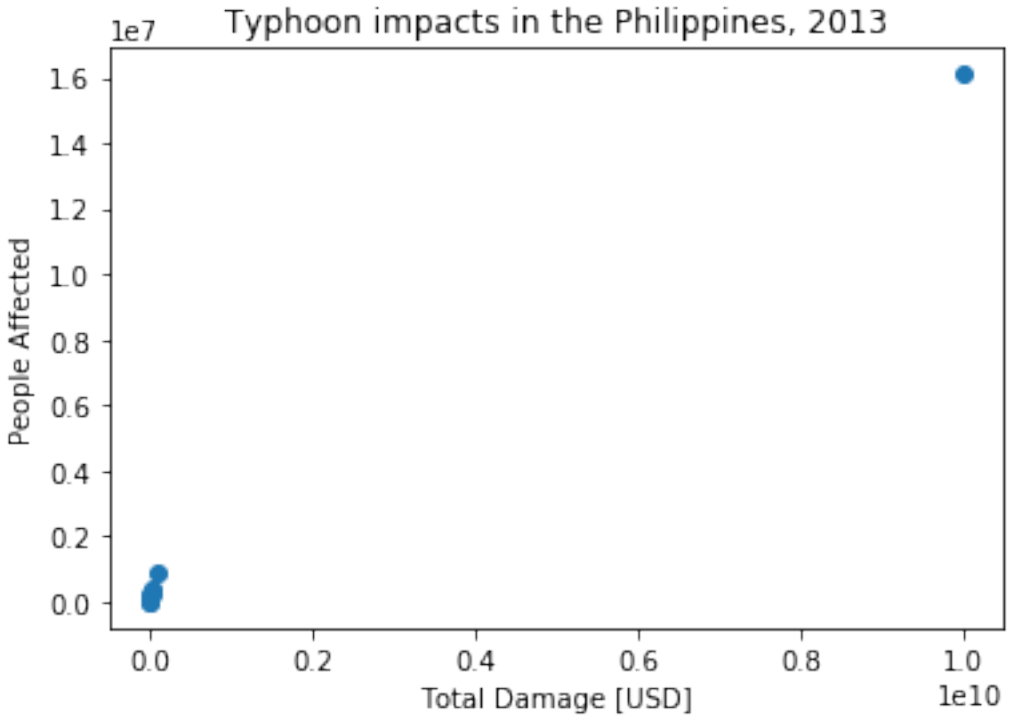
```
)

print(
    "Number of TC events in EM-DAT 2000 to 2009 globally: %i"
    % (impact_emdat.event_id.size)
)
print(
    "Global annual average monetary damage (AAI) from TCs as reported in EM-DAT 2000␣
↪to 2009: USD billion %2.2f"
    % (impact_emdat.aai_agg / 1e9)
)
```

```
2021-10-19 16:44:58,210 - climada.engine.impact_data - WARNING - ISO3alpha code not␣
↪found in iso_country: SPI
2021-10-19 16:44:59,007 - climada.engine.impact_data - WARNING - Country not found in␣
↪iso_country: SPI
Number of TC events in EM-DAT 2000 to 2009 globally: 533
Global annual average monetary damage (AAI) from TCs as reported in EM-DAT 2000 to␣
↪2009: USD billion 38.07
```

```
"""Total people affected by TCs in the Philippines in 2013:"""

# People affected
impact_emdat_PHL, countries = emdat_to_impact(
    emdat_file_path,
    "TC",
    countries="PHL",
    year_range=(2013, 2013),
    imp_str="Total Affected",
)

print(
    "Number of TC events in EM-DAT in the Philipppines, 2013: %i"
    % (impact_emdat_PHL.event_id.size)
)
print("\nPeople affected by TC events in the Philippines in 2013 (per event):")
print(impact_emdat_PHL.at_event)
print("\nPeople affected by TC events in the Philippines in 2013 (total):")
print(int(impact_emdat_PHL.aai_agg))

# Comparison to monetary damages:
impact_emdat_PHL_USD, _ = emdat_to_impact(
    emdat_file_path, "TC", countries="PHL", year_range=(2013, 2013)
)

ax = plt.scatter(impact_emdat_PHL_USD.at_event, impact_emdat_PHL.at_event)
plt.title("Typhoon impacts in the Philippines, 2013")
plt.xlabel("Total Damage [USD]")
plt.ylabel("People Affected");
# plt.xscale('log')
# plt.yscale('log')
```

```
Number of TC events in EM-DAT in the Philipppines, 2013: 8

People affected by TC events in the Philippines in 2013 (per event):
[7.269600e+04 1.059700e+04 8.717550e+05 2.204430e+05 1.610687e+07
 3.596000e+03 3.957300e+05 2.628840e+05]

People affected by TC events in the Philippines in 2013 (total):
17944571
```

```
Text(0, 0.5, 'People Affected')
```



emdat_impact_yearlysum()

function to load EM-DAT impact data and return DataFrame with impact summed per year and country

Parameters:

- emdat_file_csv (str): Full path to EMDAT-file (CSV)

Optional parameters:

- hazard (list or str): List of Disaster (sub-)type according EMDAT terminology or CLIMADA hazard type abbreviations. e.g. ['Wildfire', 'Forest fire'] or ['BF']
- year_range (list with 2 integers): start and end year e.g. [1980, 2017]
- countries (list of str): country ISO3-codes or names, e.g. ['JAM', 'CUB']. Set to None or ['all'] for all countries
- reference_year (int): reference year of exposures for normalization. Impact is scaled proportional to GDP to the value of the reference year. No scaling for reference_year=0 (default)

- imp_str (str): Column name of impact metric in EMDAT CSV, e.g. 'Total Affected'; default = "Total Damages"
- version (int): given EM-DAT data format version (i.e. year of download), changes naming of columns/variables (default: 2020)

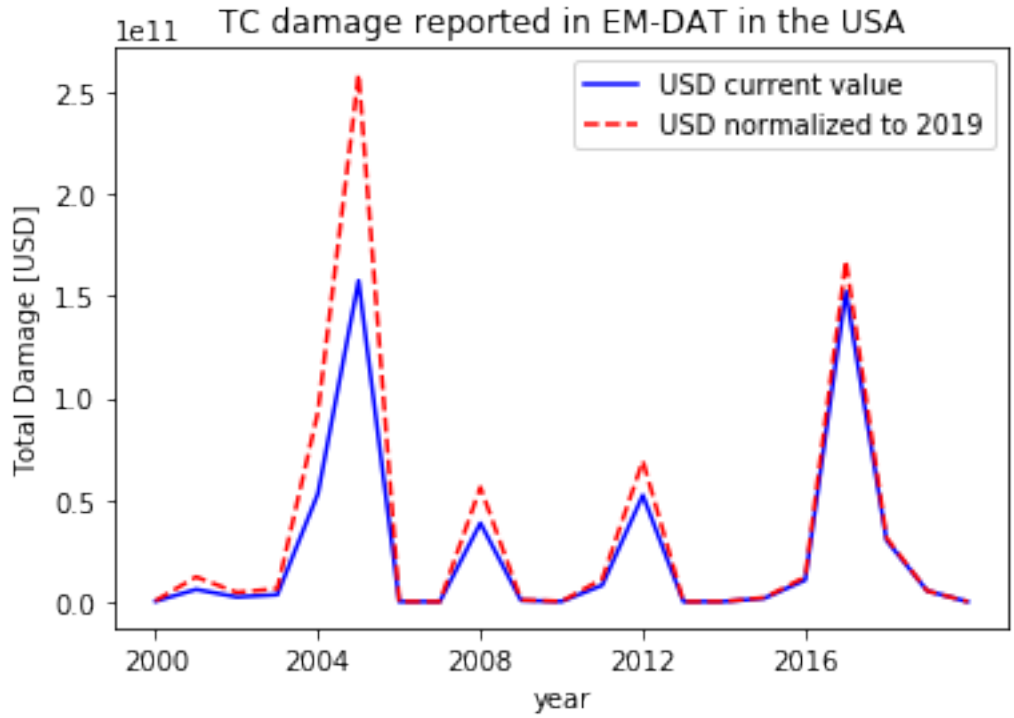
Returns:

- pandas.DataFrame with impact per year and country

```
"""Yearly TC damages in the USA, normalized and current"""

yearly_damage_normalized_to_2019 = emdat_impact_yearlysum(
    emdat_file_path,
    countries="USA",
    hazard="Tropical cyclone",
    year_range=None,
    reference_year=2019,
)

yearly_damage_current = emdat_impact_yearlysum(
    emdat_file_path,
    countries=["USA"],
    hazard="TC",
)

import matplotlib.pyplot as plt

fig, axis = plt.subplots(1, 1)
axis.plot(
    yearly_damage_current.year,
    yearly_damage_current.impact,
    "b",
    label="USD current value",
)
axis.plot(
    yearly_damage_normalized_to_2019.year,
    yearly_damage_normalized_to_2019.impact_scaled,
    "r--",
    label="USD normalized to 2019",
)
plt.legend()
axis.set_title("TC damage reported in EM-DAT in the USA")
axis.set_xticks([2000, 2004, 2008, 2012, 2016])
axis.set_xlabel("year")
axis.set_ylabel("Total Damage [USD]");
```

```
[2000 2001 2002 2003 2004 2005 2006 2007 2008 2009 2010 2011 2012 2014
 2015 2016 2017 2018 2019 2020]
```

```
Text(0, 0.5, 'Total Damage [USD]')
```



2.5.6 END-TO-END COST BENEFIT CALCULATION

Introduction

The goal of this tutorial is to show a full end-to-end cost-benefit calculation. Note that this tutorial shows the work flow and some data exploration, but does not explore all possible features.

The tutorial will start with an explanation of the mathematics of an cost-benefit calculation, and then will move on to how this is implemented in CLIMADA. We will then go through a few end-to-end calculations.

If you just need to see the code in action, you can skip these first parts, which are mostly for reference.

The tutorial assumes that you're already familiar with CLIMADA's ***Hazard***, ***Exposures***, ***impact functions***, ***Impact*** and ***adaptation measure*** functionality. The cost-benefit calculation is often the last part of an analyses, and it brings all the previous components together.

What is a cost-benefit?

A cost-benefit analysis in CLIMADA lets you compare the effectiveness of different hazard adaptation options.

The cost-benefit ratio describes how much loss you can prevent per dollar of expenditure (or whatever currency you're using) over a period of time. When a cost-benefit ratio is less than 1, the cost is less than the benefit and CLIMADA is predicting a worthwhile investment. Smaller ratios therefore represent better investments. When a cost-benefit is greater than 1, the cost is more than the benefit and the offset losses are less than the cost of the adaptation measure: based on the financials alone, the measure may not be worth it. Of course, users may have factors beyond just cost-benefits that influence decisions.

CLIMADA doesn't limit cost-benefits to just financial exposures. The cost-benefit ratio could represent hospitalisations avoided per Euro spent, or additional tons of crop yield per Swiss Franc.

The cost-benefit calculation has a few complicated components, so in this section we'll build up the calculation step by step.

Simple cost-benefits

The simplest form of a cost-benefit calculation goes like this:

$$\text{CostBenefit} = \frac{\text{cost}}{\text{benefit}} = \frac{\text{cost}}{N \cdot (\text{AAI without measures} - \text{AAI with measures})}$$

where cost is the cost of implementing a set of measures, the AAI is the average annual impact from your hazard event set on your exposure, and N is the number of years the cost-benefit is being evaluated over.

Note that:

- Whether an adaptation measure is seen to be effective might depend on the number of years you are evaluating the cost-benefit over. For example, a €50 mn investment that prevents an average of €1 mn losses per year will only 'break even' after $N = 50$ years.
- Since an adaptation measure could in theory make an impact worse (a negative benefit) it is possible to have negative cost-benefit ratios.
- CLIMADA allows you to use other statistics than annual average impact, but to keep thing simple we'll use average annual impact throughout this tutorial.

Time-dependence

The above equation works well when the only thing changing is an adaptation measure. But usually CLIMADA cost-benefit calculation will want to describe a climate and exposure that also change over time. In this case it's not enough to multiply the change in average annual impact by the number of years we're evaluating over, and we need to calculate a benefit for every year separately and sum them up.

We can modify the benefit part of cost-benefit to reflect this. CLIMADA doesn't assume that the user will have explicit hazard and impact objects for every year in the study period, and so interpolates between the impacts at the start and the end of the period of interest. If we're evaluating between years T_0, usually close to the present, and T_1 in the future, then we can say:

$$\text{benefit} = \sum_{t=T_0}^{T_1} \alpha(t) \left(\text{AAI with measures}_{T_1} - \text{AAI with measures}_{T_0} \right) - N * \text{AAI without measure}_{T_0}$$

Where $\alpha(t)$ is a function of the year t describing the interpolation of hazard and exposure values between T_0 and T_1. The function returns values in the range $[0, 1]$, usually with $\alpha(T_0) = 0$ and $\alpha(T_0) = 1$.

Note that:

- This calculation now requires three separate impact calculations: present-day impacts without measures implemented, present-day impacts with measures implemented, and future impacts with measures implemented.
- Setting $\alpha(t) = 1$ for all values of t simplifies this to the first cost-benefit equation above.

CLIMADA lets you set $\alpha(t)$ to 1 for all years t, or as $\$\alpha_k(t) = \frac{(t-T_0)^k}{(T_1-T_0)^k}$ for $t \in [T_0, T_1]\$$

where k is user-provided, called `imp_time_depen`. This expression is a polynomial curve between T_0 and T_1 normalised so that $\alpha_k(T_0) = 0$ and $\alpha_k(T_1) = 1$. The choice of k determines how quickly the transition occurs between the present and future. When $k = 1$ the function is a straight line. When $k > 1$ change begins slowly and speeds up over time. When $k < 1$ change is begins quickly and slows over time.

If this math is tough, the key takeaways are

- Cost benefit calculations take a long view, summing the benefits of adaptation measures over many years in a changing world

- CLIMADA describes how the change from the present to the future scenarios happens with the `imp_time_depen` parameter. With values < 1 the change starts quickly and slows down. When it is equal to 1 change is steady. When it's > 1 change starts slowly and speeds up.

Discount rates

The final addition to our cost-benefit calculation is the *discount rate*.

The discount rate tries to formalise an idea from economics that says that a gain in the future is worth less to us than the same gain right now. For example, paying €1 mn to offset €2 mn of economic losses next year is 'worth more' than paying €1 mn to offset €2 mn of economic losses in 2080.

In practice it provides a way to future monetary values to an estimated worth today, called their *net present value*. Note that this is *not* an adjustment for inflation.

The choice of discount rate is a contentious topic in adaptation finance, since it can strongly affect a cost-benefit calculation. The most widley used discount rate in climate change economics is 1.4% as proposed by the Stern Review (2006). Neoliberal economists around Nordhaus (2007) claim that rates should be higher, around 4.3%, reflecting continued economic growth and a society that will be better at adapting in the future compared to now. Environmental economists argue that future costs shouldn't be discounted at all.

To illustrate, with a 1.4% annual discount rate, a gain of €100 next year is equivalent to €98.60 this year, and a gain of €100 in 15 years is equivalent to €$(100 * 0.986^{15})$ = € 80.94 this year. With a rate of 4.3% this drops to €51.72.

We can add this into the cost-benefit calculation by defining $d(t)$, the discount rate for each year t. A constant rate of 1.4% would then set $d(t) = 0.014$ for all values of t.

Then the adjustment $D(t)$ from year t to the net present value in year T_0 is given by

$$D(t) = \prod_{y=T_0}^{t} (1 - d(y))$$

With a constant 1.4% discount rate, we have $D(t) = 0.986^{t-T_0}$. With a discount rate of zero we have $D(t) = 1$.

Adding this to our equation for total benefits we get:

$$\text{benefit} = \sum_{t=T_0}^{T_1} \alpha(t) D(t) (\text{AAI with measures}_{T_1} - \text{AAI with measures}_{T_0}) - N * \text{AAI without measure}_{T_0}$$

Note:

- Setting the rates to zero ($d(t) = 0$) means $D(t) = 1$ and the term drops out of the equation.
- Be careful with your choice of discount rate when your exposure is non-economic. It can be hard to justify applying rates to e.g. ecosystems or human lives.

CostBenefit class data structure

The CostBenefit class does not require any attributes to be defined by the user. All attributes are set from parameters when the method `CostBenefit.calc()` is called.

After calling the `calc` method the `CostBenefit` object has the following attributes:

Attributes created in `CostBenefit.calc`	Data Type	Description
present_year	int	The current year
future_year	int	The future scenario year
tot_climate_risk	float	The total climate risk in the present scenario, evaluated according to the provided risk function (annual average impact by default)
unit	string	Units to measure impact
benefit	dict(flo	The benefit of each measure, keyed by measure name
cost_ben_ratio	dict(flo	The cost benefit of each measure, keyed by measure name
imp_meas_future	dict(di	Dictionaries describing the impacts of each measure in the future scenario. Keyed by measure name (with 'no measure' for no measures). The entries in each dictionary are described below.
imp_meas_present	dict(di	Dictionaries describing the impacts of each measure in the present-day scenario. Keyed by measure name (with 'no measure' for no measures). The entries in each dictionary are described below.

Each dictionary stored in the attributes `imp_meas_future` and `imp_meas_present` has entries:

Key	Data Type	Description
cost	tuple (cost measure, cost factor insurance)	The cost of implementing the measure, and the cost factor if risk transfers are being calculated
impact	Impact	Impact object calculated with the present (`imp_meas_present`) or future (`imp_meas_future`) hazard, exposure and impact functions
risk	float	A value of annual risk used in the cost-benefit calculation. A summary statistic calculated from the Impact object. Most commonly the average annual impact, but can be changed with the `CostBenefit.calc`'s `risk_func`parameter.
risk_t	float	Annual expected risk transfer (if calculated)
efc	ImpactFreqCurve	The impact exceedance freq for this measure calculated from the Impact object (if calculated)

The dictionary will also include a 'no measure' entry with the same structure, giving the impact analysis when no measures are implemented.

The `calc` calculation

Let's look at the parameters needed for the calculation:

```
CostBenefit.calc(hazard, entity, haz_future=None, ent_future=None,
                 future_year=None, risk_func=risk_aai_agg, imp_time_depen=None, save_
↪imp=False)
```

These are:

- `hazard` (Hazard object): the present-day or baseline hazard event set
- `entity` (Entity object): the present-day or baseline Entity object. `Entity` is the container class containing
  - `exposure` (Exposures object): the present-day or baseline exposure
  - `disc_rates` (DiscRates object): the discount rates to be applied in the cost-benefit calculation. Only discount rates from `entity` and not `ent_future` are used.

  - `impact_funcs` (ImpactFuncSet object): the impact functions required to calculate impacts from the present-day hazards and exposures
  - `measures` (MeasureSet object): the set of measures to implement in the analysis. This will almost always be the same as the measures in the `ent_future` Entity (if set).

- `haz_future` (Hazard object, optional): the future hazard event set, if different from present.
- `ent_future` (Entity object, optional): the future Entity, if different from present. Note that the same adaptation measures must be present in both `entity` and `ent_future`.
- `future_year` (int): the year of the future scenario. This is only used if the Entity's `exposures.ref_year` isn't set, or no future entity is provided.
- `risk_func` (function): this is the risk function used to describe the annual impacts used to describe benefits. The default is `risk_aai_agg`, the average annual impact on the Exposures (defined in the CostBenefit module). This function can be replaces with any function that takes an Impact object as input and returns a number. The CostBenefit module provides two others functions `risk_rp_100` and `risk_rp_250`, the 100-year and 250-year return period impacts respectively.
- `imp_time_depen` (float): This describes how hazard and exposure evolve over time in the calculation. In the descriptions above this is the parameter k defining $\alpha_k(t)$. When > 1 change is superlinear and occurs nearer the start of the analysis. When < 1 change is sublinear and occurs nearer the end.
- `save_imp` (boolean): whether to save the hazard- and location-specific impact data. This is used in a lot of follow-on calculations, but is very large if you don't need it.

Detailed CostBenefit calculation: LitPop + TropCyclone

We present a detailed example for the hazard ***Tropical Cyclones*** and the exposures from ***LitPop*** .

To speed things up we'll use the CLIMADA Data API to download the data as needed. The data download roughly follows the ***Data API tutorial*** for Haiti. If this is a problem you can build tropical cyclone event sets, and LitPop exposures following the relevant tutorials.

Download hazard

We will get data for present day tropical cyclone hazard in Haiti, and for 2080 hazard under the RCP 6.0 warming scenario. Note that the Data API provides us with a full event set of wind footprints rather than a TCTracks track dataset, meaning we don't have to generate the wind fields ourselves.

```
from climada.util.api_client import Client

client = Client()
future_year = 2080
haz_present = client.get_hazard(
    "tropical_cyclone",
    properties={
        "country_name": "Haiti",
        "climate_scenario": "historical",
        "nb_synth_tracks": "10",
    },
)
haz_future = client.get_hazard(
    "tropical_cyclone",
    properties={
        "country_name": "Haiti",
        "climate_scenario": "rcp60",
```

(continues on next page)

(continued from previous page)

```
        "ref_year": str(future_year),
        "nb_synth_tracks": "10",
    },
)
```

```
2022-03-03 05:35:22,192 - climada.hazard.base - INFO - Reading /Users/chrisfairless/
↪climada/data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_150arcsec_HTI_
↪1980_2020/v1/tropical_cyclone_10synth_tracks_150arcsec_HTI_1980_2020.hdf5
2022-03-03 05:35:28,402 - climada.hazard.base - INFO - Reading /Users/chrisfairless/
↪climada/data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_150arcsec_
↪rcp85_HTI_2080/v1/tropical_cyclone_10synth_tracks_150arcsec_rcp85_HTI_2080.hdf5
```

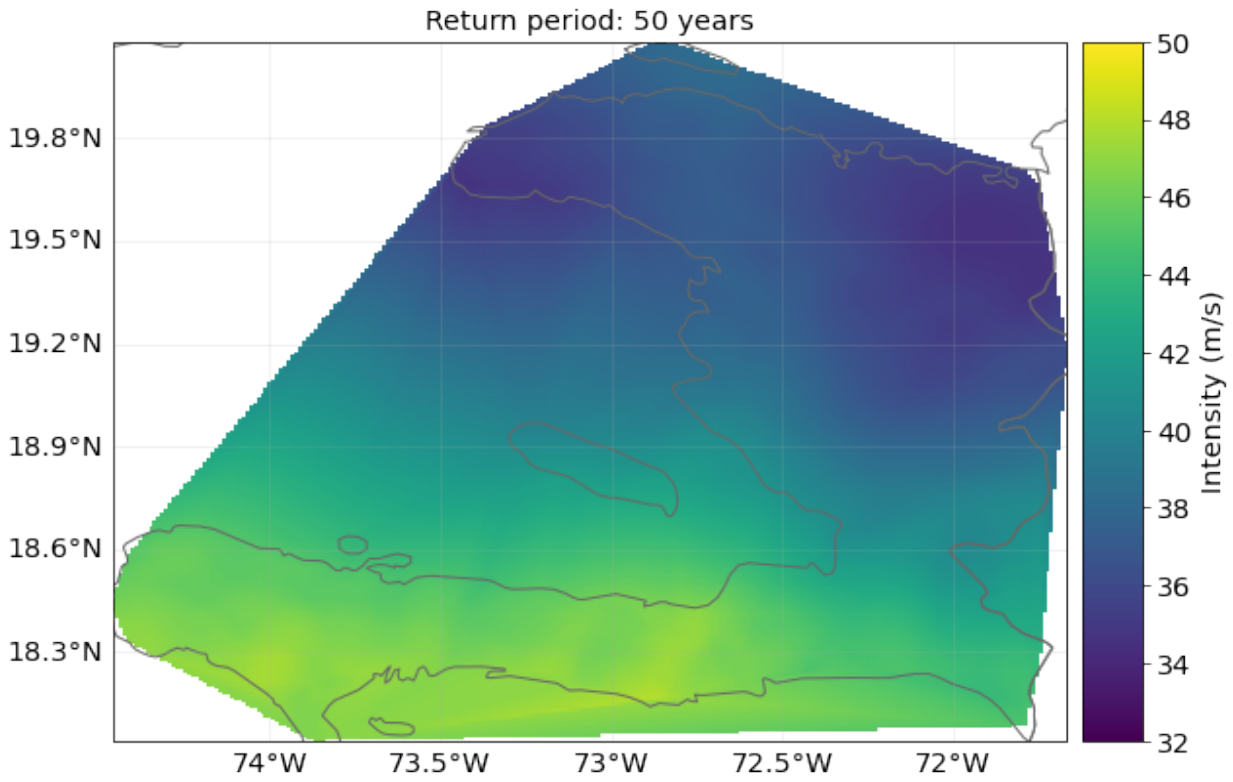
We can plot the hazards and show how they are forecast to intensify. For example, showing the strength of a 50-year return period wind in present and future climates:

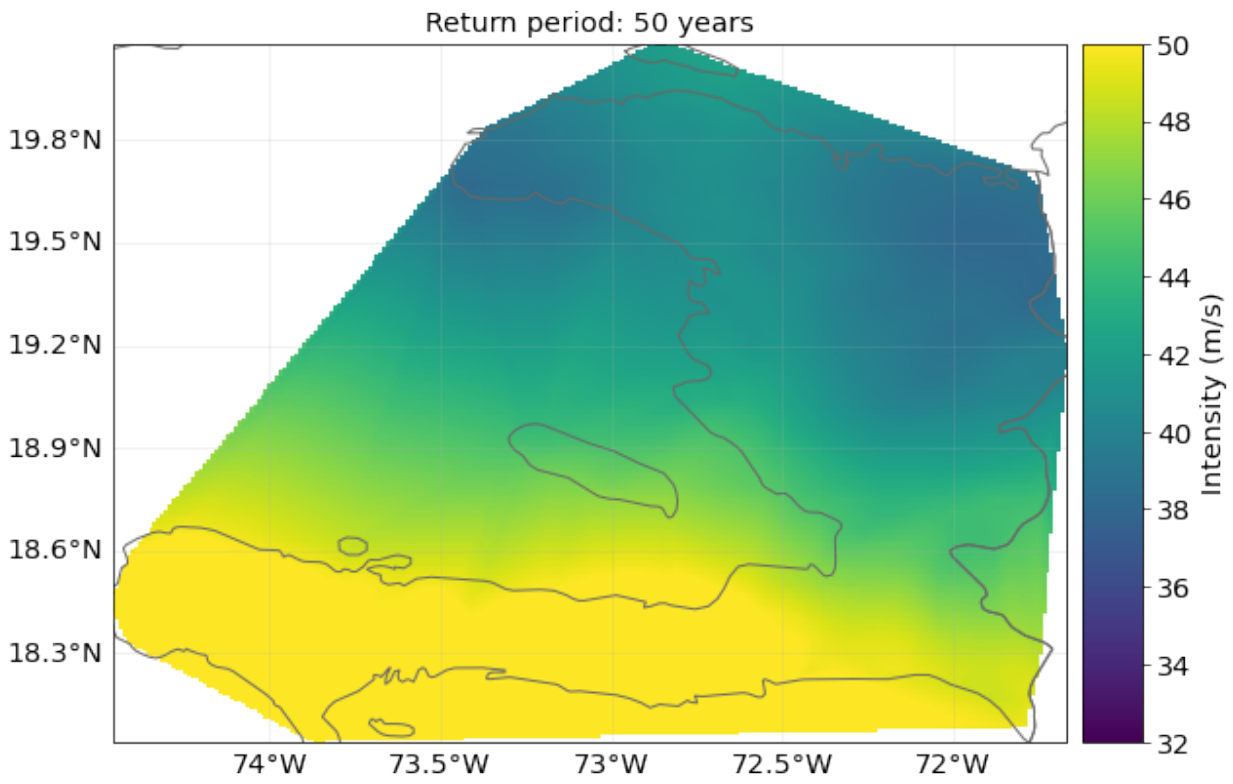
```
# Plot the hazards, showing 50-year return period hazard
haz_present.plot_rp_intensity(return_periods=(50,), smooth=False, vmin=32, vmax=50)
haz_future.plot_rp_intensity(return_periods=(50,), smooth=False, vmin=32, vmax=50)
```

```
2022-03-03 05:35:28,479 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [50]
2022-03-03 05:35:36,986 - climada.hazard.base - INFO - Computing exceedance intenstiy␣
↪map for return periods: [50]
```

```
(<GeoAxesSubplot:title={'center':'Return period: 50 years'}>,
 array([[41.84896948, 41.98439726, 41.62016887, ..., 49.52344953,
         51.35294266, 51.51945831]]))
```

```
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries␣
↪is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to␣
↪access the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```


Return period: 50 years
19.8°N
19.5°N
19.2°N
18.9°N
18.6°N
18.3°N
74°W
73.5°W
73°W
72.5°W
72°W
50
48
46
44
42
40
38
36
34
32
Intensity (m/s)


Return period: 50 years
19.8°N
19.5°N
19.2°N
18.9°N
18.6°N
18.3°N
74°W
73.5°W
73°W
72.5°W
72°W
50
48
46
44
42
40
38
36
34
32
Intensity (m/s)

Download LitPop economic exposure data

The Data API provides us with economic exposure data:

```
exp_present = client.get_litpop(country="Haiti")
```

```
2022-03-03 05:35:52,700 - climada.entity.exposures.base - INFO - Reading /Users/
↪chrisfairless/climada/data/exposures/litpop/LitPop_150arcsec_HTI/v1/LitPop_
↪150arcsec_HTI.hdf5
```

For 2080's economic exposure we will use a crude approximation, assuming the country will experience 2% economic growth annually:

```
import copy

exp_future = copy.deepcopy(exp_present)
exp_future.ref_year = future_year
n_years = exp_future.ref_year - exp_present.ref_year + 1
growth_rate = 1.02
growth = growth_rate**n_years
exp_future.gdf["value"] = exp_future.gdf["value"] * growth
```

We can plot the current and future exposures. The default scale is logarithmic and we see how the values of exposures grow, though not by a full order of magnitude.

```
exp_present.plot_raster(fill=False, vmin=4, vmax=11)
exp_future.plot_raster(fill=False, vmin=4, vmax=11)
```

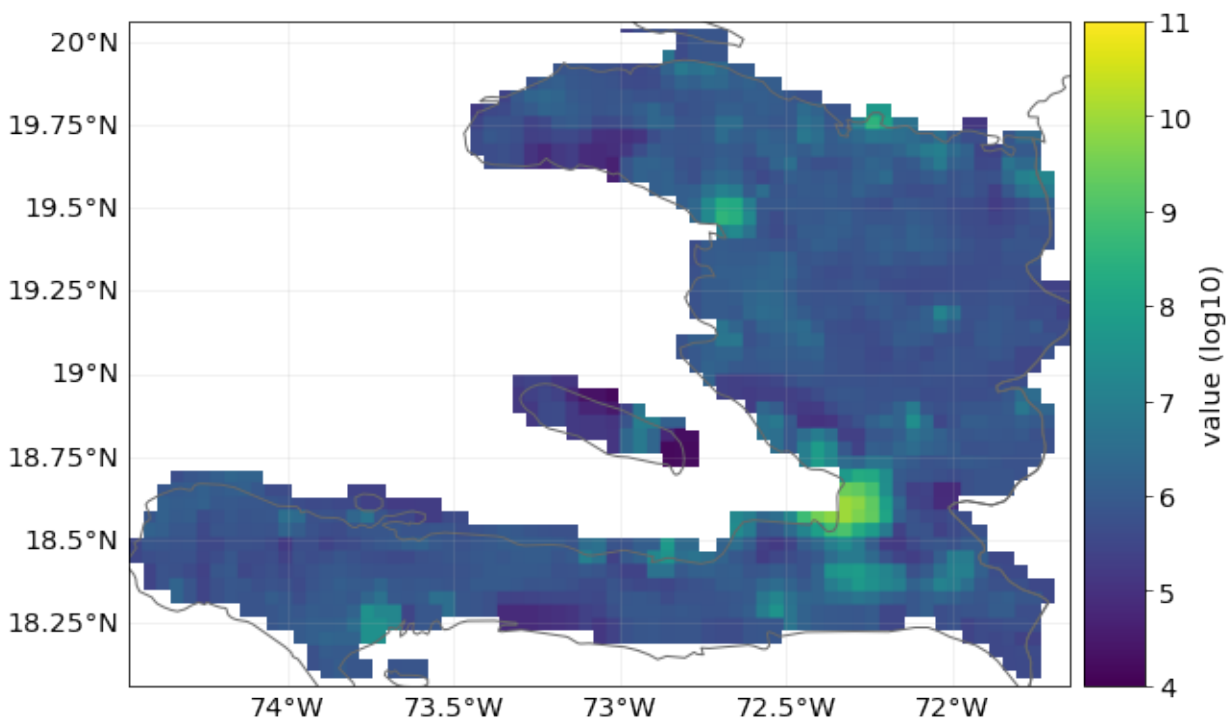
```
2022-03-03 05:35:52,895 - climada.util.coordinates - INFO - Raster from resolution 0.
↪04166666666666785 to 0.04166666666666785.
```

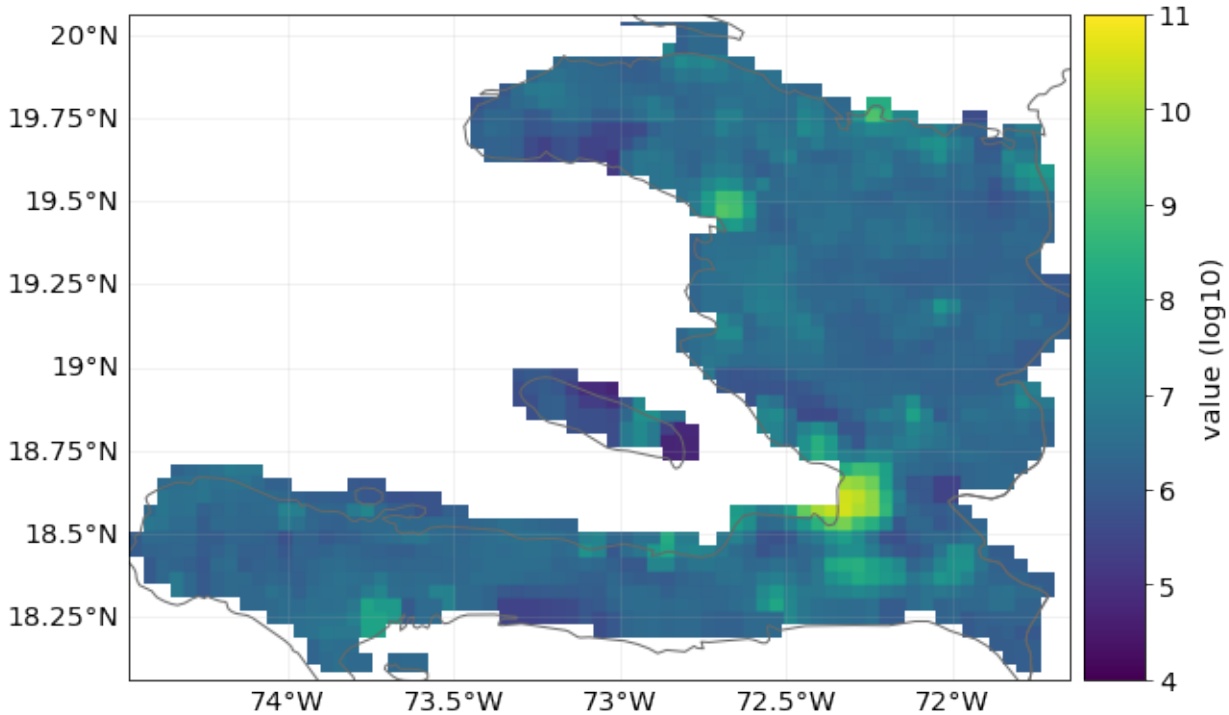
```
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪pyproj/crs/crs.py:1256: UserWarning: You will likely lose important projection␣
↪information when converting to a PROJ string from another format. See: https://proj.
↪org/faq.html#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries␣
↪is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to␣
↪access the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```

```
2022-03-03 05:35:59,536 - climada.util.coordinates - INFO - Raster from resolution 0.
↪04166666666666785 to 0.04166666666666785.
```

```
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪pyproj/crs/crs.py:1256: UserWarning: You will likely lose important projection␣
↪information when converting to a PROJ string from another format. See: https://proj.
↪org/faq.html#what-is-the-best-format-for-describing-coordinate-reference-systems
  return self._crs.to_proj4(version=version)
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:825: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(multi_line_string) > 1:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:877: ShapelyDeprecationWarning: Iteration over multi-part geometries␣
↪is deprecated and will be removed in Shapely 2.0. Use the `geoms` property to␣
↪access the constituent parts of a multi-part geometry.
  for line in multi_line_string:
/Users/chrisfairless/opt/anaconda3/envs/climada_env/lib/python3.8/site-packages/
↪cartopy/crs.py:944: ShapelyDeprecationWarning: __len__ for multi-part geometries is␣
↪deprecated and will be removed in Shapely 2.0. Check the length of the `geoms`␣
↪property instead to get the  number of parts of a multi-part geometry.
  if len(p_mline) > 0:
```

```
<GeoAxesSubplot:>
```





We then need to map the exposure points to the hazard centroids. (Note: we could have done this earlier before we copied the exposure, but not all analyses will have present and future exposures and hazards on the same sets of points.)

```
# This would be done automatically in Impact calculations
# but it's better to do it explicitly before the calculation
exp_present.assign_centroids(haz_present, distance="approx")
exp_future.assign_centroids(haz_future, distance="approx")
```

```
2022-03-03 05:36:15,405 - climada.entity.exposures.base - INFO - Matching 1329␣
↪exposures with 1329 centroids.
2022-03-03 05:36:15,421 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-03-03 05:36:16,026 - climada.entity.exposures.base - INFO - Matching 1329␣
↪exposures with 1329 centroids.
2022-03-03 05:36:16,042 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
```

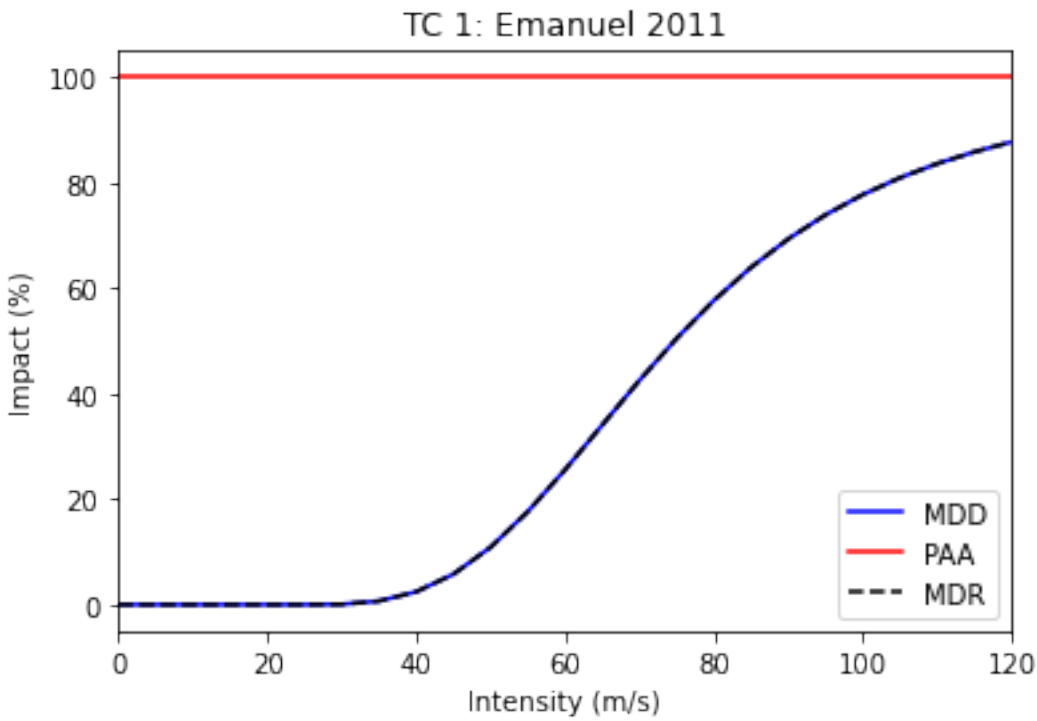
Define impact function

In this analysis we'll use the popular sigmoid curve impact function from Emanuel (2011).

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone

impf_tc = ImpfTropCyclone.from_emanuel_usa()

# add the impact function to an Impact function set
impf_set = ImpactFuncSet([impf_tc])
impf_set.check()
impf_tc.plot()
```

```
2022-03-03 05:36:16,069 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```

```
<AxesSubplot:title={'center':'TC 1: Emanuel 2011'}, xlabel='Intensity (m/s)', ylabel=
↪'Impact (%)'>
```




```
# Rename the impact function column in the exposures and assign hazard IDs

# This is more out of politeness, since if there's only one impact function
# and one `impf_` column, CLIMADA can figure it out
exp_present.gdf.rename(columns={"impf_": "impf_TC"}, inplace=True)
exp_present.gdf["impf_TC"] = 1
exp_future.gdf.rename(columns={"impf_": "impf_TC"}, inplace=True)
exp_future.gdf["impf_TC"] = 1
```

Define adaptation measures

For adaptation measures we'll follow some of the examples from the *Adaptation MeasureSet tutorial*. See the tutorial to understand how measures work in more depth.

These numbers are completely made up. We implement one measure that reduces the (effective) wind speed by 5 m/s and one that completely protects 10% of exposed assets.

```
import numpy as np
import matplotlib.pyplot as plt
from climada.entity.measures import Measure, MeasureSet
```

(continues on next page)

(continued from previous page)

```
meas_1 = Measure(
    haz_type="TC",
    name="Measure A",
    color_rgb=np.array([0.8, 0.1, 0.1]),
    cost=5000000000,
    hazard_inten_imp=(1, -5),  # Decrease wind speeds by 5 m/s
    risk_transf_cover=0,
)

meas_2 = Measure(
    haz_type="TC",
    name="Measure B",
    color_rgb=np.array([0.1, 0.1, 0.8]),
    cost=220000000,
    paa_impact=(1, -0.10),  # 10% fewer assets affected
)

# gather all measures
meas_set = MeasureSet(measure_list=[meas_1, meas_2])
meas_set.check()
```

Define discount rates

We'll define two discount rate objects so that we can compare their effect on a cost-benefit. First, a zero discount rate, where preventing loss in 2080 is valued the same a preventing it this year. Second, the often-used 1.4% per year.

```
from climada.entity import DiscRates

year_range = np.arange(exp_present.ref_year, exp_future.ref_year + 1)
annual_discount_zero = np.zeros(n_years)
annual_discount_stern = np.ones(n_years) * 0.014

discount_zero = DiscRates(year_range, annual_discount_zero)
discount_stern = DiscRates(year_range, annual_discount_stern)
```

Create Entity objects

Now we have everything we need to create Entities. Remember, Entity is a container class for grouping ***Exposures***, ***Impact Functions***, ***Discount Rates*** and ***Measures***.

In this first example we'll set discount rates to zero.

```
from climada.entity import Entity

entity_present = Entity(
    exposures=exp_present,
    disc_rates=discount_zero,
    impact_func_set=impf_set,
    measure_set=meas_set,
)
entity_future = Entity(
    exposures=exp_future,
```

(continues on next page)

(continued from previous page)

```
    disc_rates=discount_zero,
    impact_func_set=impf_set,
    measure_set=meas_set,
)
```

Cost-benefit #1: adaptation measures, no climate change or economic growth

We are now ready to perform our first cost-benefit analysis. We'll start with the simplest and build up complexity.

The first analysis only looks at solely at the effects of introducing adaptation measures. It assumes no climate change and no economic growth. It evaluates the benefit over the period 2018 (present) to 2080 (future) and sets the discount rate to zero.

```
from climada.engine import CostBenefit
from climada.engine.cost_benefit import risk_aai_agg

costben_measures_only = CostBenefit()
costben_measures_only.calc(
    haz_present,
    entity_present,
    haz_future=None,
    ent_future=None,
    future_year=future_year,
    risk_func=risk_aai_agg,
    imp_time_depen=None,
    save_imp=True,
)
```

```
2022-03-03 05:36:16,236 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,238 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,258 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,259 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,295 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,296 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,332 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2080.

Measure      Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
---------  ---------------  ------------------  --------------
Measure A             5                4.74132        0.948265
Measure B             0.22             1.10613        5.02787

--------------------  ---------  --------
Total climate risk:   11.0613    (USD bn)
Average annual risk:   0.175576  (USD bn)
Residual risk:         5.21385   (USD bn)
```

(continues on next page)

(continued from previous page)

```
--------------------  ---------  --------
Net Present Values
```

Let's take a moment to look through these results.

The first table gives us a breakdown of cost-benefits by measure. We can see that, the Benefit/Cost for measure A is very just under 1, meaning that the damage prevented is slightly less than the cost of preventing it (according to the model). In comparison, the benefit of Measure B is 5 times the cost. (Note that Benefit/Cost is the inverse of Cost/Benefit: larger numbers are better).

Let's explain the three values in the second table:

- **Total climate risk:** The impact expected over the entire study period. With no changes in future hazard or exposure and no discount rates we can check that it is 63 times the next term.
- **Average annual risk:** The average annual risk without any measures implemented in the future scenario (which here is the same as the present day scenario)
- **Residual risk:** The remaining risk that hasn't been offset by the adaptation measures. Here it is the total climate risk minus the total of the 'Benefit' column of the table above it.

Combining measures

We can also combine the measures to give the cost-benefit of implementing everything:

```
combined_costben = costben_measures_only.combine_measures(
    ["Measure A", "Measure B"],
    "Combined measures",
    new_color=np.array([0.1, 0.8, 0.8]),
    disc_rates=discount_zero,
)
```

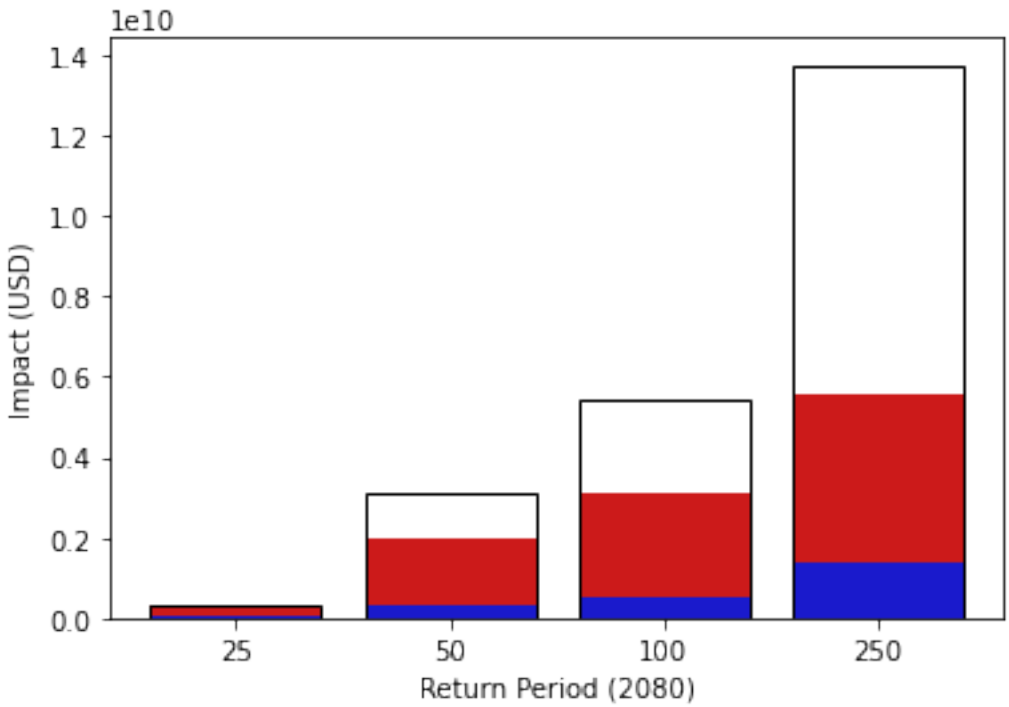
```
Measure               Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Combined measures             5.22             5.84344         1.11943

--------------------  ---------  --------
Total climate risk:   11.0613    (USD bn)
Average annual risk:   0.175576  (USD bn)
Residual risk:         5.21787   (USD bn)
--------------------  ---------  --------
Net Present Values
```

Note: the method of combining measures is naive. The offset impacts are summed over the event set while not letting the impact of any single event drop below zero (it therefore doesn't work in analyses where impacts can go below zero).

Plotting benefits by return period

Finally, we can see how effective the adaptation measures are at different return periods. The `plot_event_view` plot shows the difference in losses at different return periods in the future scenario (here the same as the present scenario) with the losses offset by the adaptation measures shaded.

```
ax = costben_measures_only.plot_event_view((25, 50, 100, 250))
```



We see that the Measure A, which reduces wind speeds by 5 m/s, is able to completely stop impacts at the 25 year return period, and that at 250 years – the strongest events – the measures have greatly reduced effectiveness.

Cost-benefit #2: adaptation measures with climate change and economic growth

Our next analysis will introduce a change in future scenarios. We'll add `hazard_future` and `entity_future` into the mixture. We'll set `imp_time_depen` set to 1, meaning we interpolate linearly between the present and future hazard and exposures in our summation over years. We'll still keep the discount rate at zero.

```
costben = CostBenefit()
costben.calc(
    haz_present,
    entity_present,
    haz_future=haz_future,
    ent_future=entity_future,
    future_year=future_year,
    risk_func=risk_aai_agg,
    imp_time_depen=1,
    save_imp=True,
)
```

```
2022-03-03 05:36:16,478 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,480 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,498 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,500 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,534 - climada.engine.impact - INFO - Exposures matching centroids␣
```

(continues on next page)

(continued from previous page)

```
↪found in centr_TC
2022-03-03 05:36:16,535 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,572 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,574 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,592 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,593 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,615 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,616 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,637 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2080.

Measure      Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
---------  ---------------  ------------------  --------------
Measure A             5               13.9728          2.79457
Measure B             0.22             3.65387        16.6085

--------------------  ---------  --------
Total climate risk:   36.5387    (USD bn)
Average annual risk:   0.984382  (USD bn)
Residual risk:        18.912     (USD bn)
--------------------  ---------  --------
Net Present Values
```

What has changed by adding climate change and population growth?

- With growing exposure and more extreme events we see about a 3-fold increase in total climate risk. Remember this is the average annual impacts summed over every year between the present and the future. The average annual risk has grown even more, by over a factor of 5. This represents the unadapted annual impacts in the future scenario.
- Greater impacts means that our adaptation measures offset more in absolute terms. The same adaptation measures create larger benefits (and therefore cost-benefits), which have increased by about a factor of three. Measure A is now clearly worth implementing in this cost-benefit analysis, whereas it wasn't before.
- We also see that the residual risk, i.e. the impacts over the analysis period that are not offset by the adaptation measures, is much larger.

Exercise: try changing the value of the `imp_time_depen` parameter in the calculation above. Values < 1 front-load the changes over time, and values > 1 back-load the changes. How does it affect the values in the printout above? What changes? What doesn't?

Waterfall plots

Now that there are more additional components in the analysis, we can use more of the CostBenefit class's visualisation methods. The waterfall plot is the clearest way to break down the components of risk:

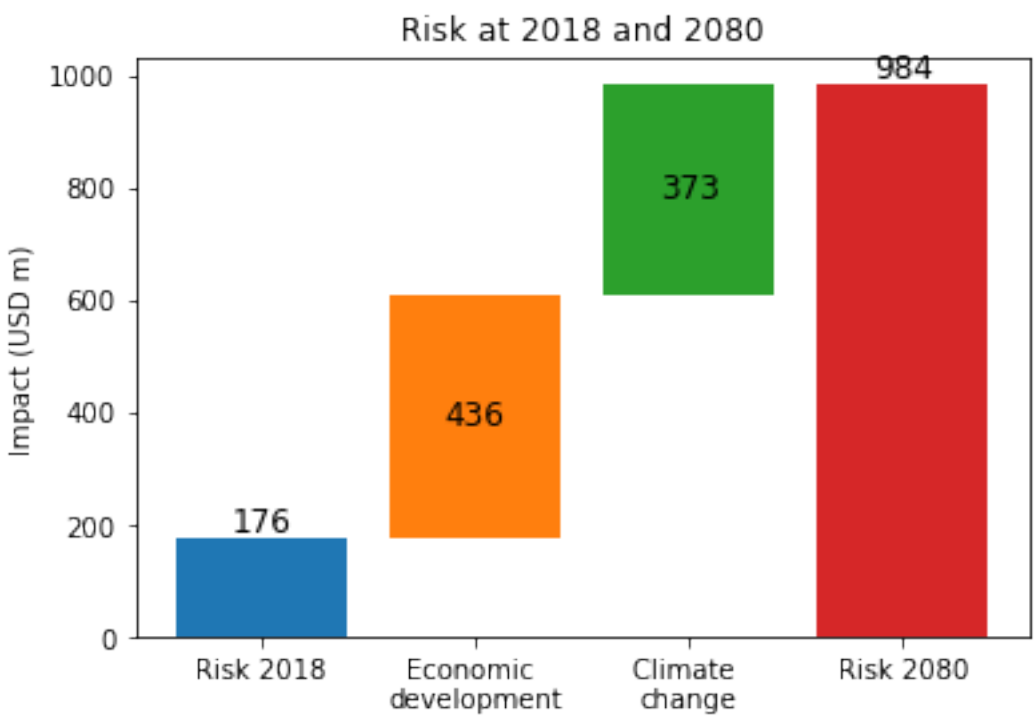
```
# define this as a function because we'll use it again later
def waterfall():
```

(continues on next page)

(continued from previous page)

```
    return costben.plot_waterfall(
        haz_present, entity_present, haz_future, entity_future, risk_func=risk_aai_agg
    )


ax = waterfall()
```

```
2022-03-03 05:36:16,647 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,649 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,665 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,667 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,695 - climada.engine.cost_benefit - INFO - Risk at 2018: 1.756e+08
2022-03-03 05:36:16,696 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,699 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,714 - climada.engine.cost_benefit - INFO - Risk with development␣
↪at 2080: 6.113e+08
2022-03-03 05:36:16,715 - climada.engine.cost_benefit - INFO - Risk with development␣
↪and climate change at 2080: 9.844e+08
```



The waterfall plot breaks down the average annual risk faced in 2080 (this is \$0.984 bn, as printed out during the cost-benefit calculation).

We see that the baseline 2018 risk in blue. The 'Economic development' bar in orange shows the change in annual impacts

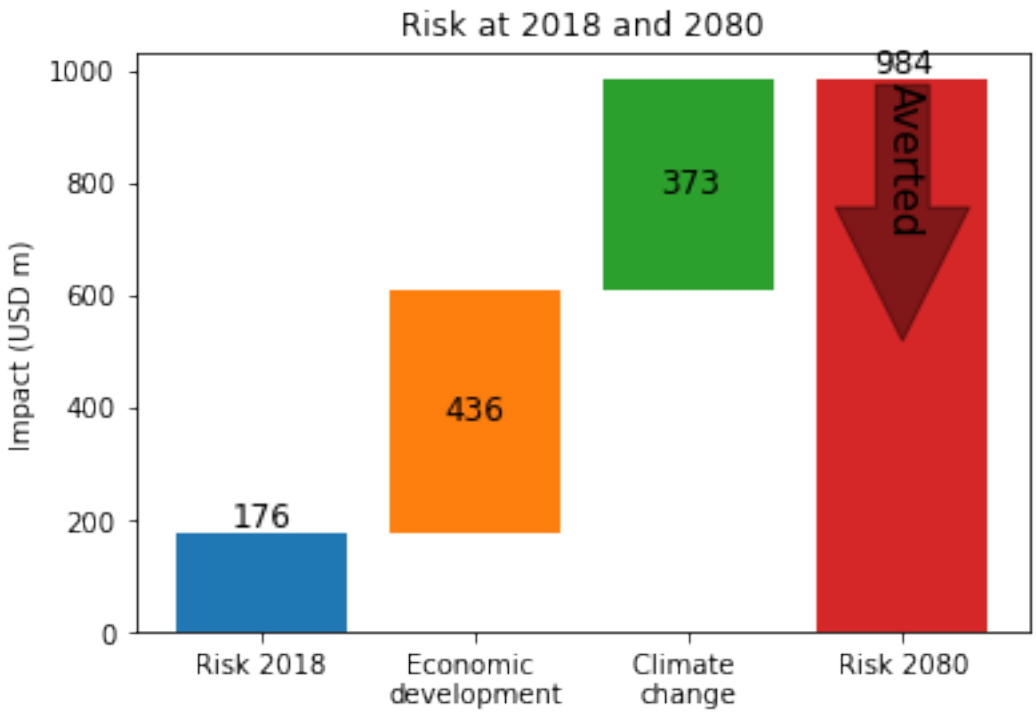
resulting from growth in exposure, and the 'Climate change' bar in green shows the additional change from changes in the hazard.

In this analysis, then, we see that changes in annual losses are likely to be driven by both economic development and climate change, in roughly equal amounts.

The `plot_arrow_averted` graph builds on this, adding an indication of the risk averted to the waterfall plot. It's slightly awkward to use, which is why we wrote a function to create the waterfall plot earlier:

```
costben.plot_arrow_averted(
    axis=waterfall(),
    in_meas_names=["Measure A", "Measure B"],
    accumulate=True,
    combine=False,
    risk_func=risk_aai_agg,
    disc_rates=None,
    imp_time_depen=1,
)
```

```
2022-03-03 05:36:16,803 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,804 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,820 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,821 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:16,849 - climada.engine.cost_benefit - INFO - Risk at 2018: 1.756e+08
2022-03-03 05:36:16,850 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,851 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,866 - climada.engine.cost_benefit - INFO - Risk with development␣
↪at 2080: 6.113e+08
2022-03-03 05:36:16,867 - climada.engine.cost_benefit - INFO - Risk with development␣
↪and climate change at 2080: 9.844e+08
```



Exercise: In addition, the `plot_waterfall_accumulated` method is available to produce a waterfall plot from a different perspective. Instead of showing a breakdown of the impacts from the year of our future scenario, it accumulates the components of risk over the whole analysis period. That is, it sums the components over every year between 2018 (when the entire risk is the baseline risk) to 2080 (when the breakdown is the same as the plot above). The final plot has the same four components, but gives them different weightings. Look up the function in the `climada.engine.cost_benefit` module and try it out. Then try changing the value of the `imp_time_depen` parameter, and see how front-loading or back-loading the year-on-year changes gives different totals and different breakdowns of risk.

Cost-benefit #3: Adding discount rates

Next we will introduce discount rates to the calculations. Recall that discount rates are factors used to convert future impacts into present-day impacts, based on the idea that an impact in the future is less significant than the same impact today.

We will work with the annual 1.4% discount that we defined earlier in the `discount_stern` object. Let's define two new Entity objects with these discount rates:

```
entity_present_disc = Entity(
    exposures=exp_present,
    disc_rates=discount_stern,
    impact_func_set=impf_set,
    measure_set=meas_set,
)
entity_future_disc = Entity(
    exposures=exp_future,
    disc_rates=discount_stern,
    impact_func_set=impf_set,
    measure_set=meas_set,
)
```

And then re-calculate the cost-benefits:

```
costben_disc = CostBenefit()
costben_disc.calc(
    haz_present,
    entity_present_disc,
    haz_future=haz_future,
    ent_future=entity_future_disc,
    future_year=future_year,
    risk_func=risk_aai_agg,
    imp_time_depen=1,
    save_imp=True,
)
print(costben_disc.imp_meas_future["no measure"]["impact"].imp_mat.shape)
```

```
2022-03-03 05:36:16,969 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,971 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:16,988 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:16,989 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:17,024 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:17,026 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 42779 events.
2022-03-03 05:36:17,062 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:17,064 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:17,079 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:17,081 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:17,100 - climada.engine.impact - INFO - Exposures matching centroids␣
↪found in centr_TC
2022-03-03 05:36:17,101 - climada.engine.impact - INFO - Calculating damage for 1329␣
↪assets (>0) and 16808 events.
2022-03-03 05:36:17,123 - climada.engine.cost_benefit - INFO - Computing cost benefit␣
↪from years 2018 to 2080.

Measure      Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
---------  ---------------  ------------------  --------------
Measure A             5                8.46661         1.69332
Measure B             0.22             2.20086        10.0039

--------------------  ---------  --------
Total climate risk:   22.0086    (USD bn)
Average annual risk:   0.984382  (USD bn)
Residual risk:        11.3412    (USD bn)
--------------------  ---------  --------
Net Present Values
(0, 0)
```

How has this changed the numbers?

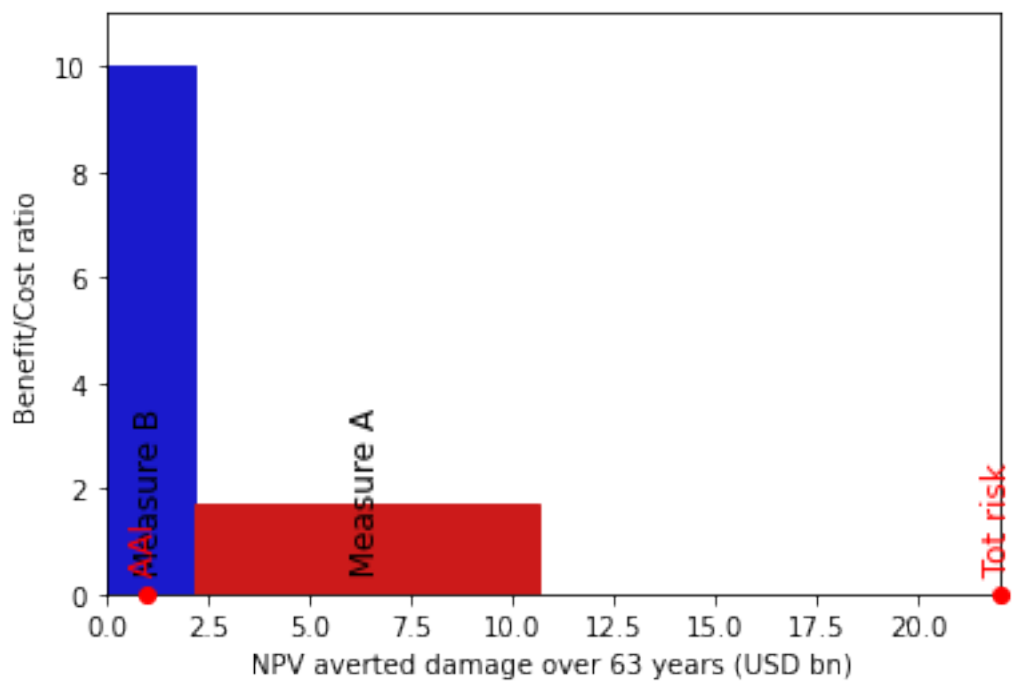
- The benefits have shrunk, since the values of the impacts prevented in the future have decreased. This means the benefit/cost ratios have decreased too. Nevertheless Measure A still has benefits that outweigh the costs.
- The total climate risk has decreased. The risk is the sum of impacts (with no adaptation measures) over the whole analysis period. Since future impacts are all smaller than they were without discount rates, the sum has decreased.
- The average annual risk, the unadapted annual risk in the future scenario, is the same (although it would be less if it was converted to a net present value).
- The residual risk has also shrunk. It has been discounted in the same way as the offset impacts.

Take together we see a slightly more optimistic outlook for climate change, as the future risks are smaller, but less attractive investments in offsetting these risks, since the benefit/cost ratio has shrunk.

Additional data exploration

With scenarios like this, the `CostBenefit.plot_cost_benefit` method shows a 2-dimensional representation of the cost-benefits.

```
ax = costben_disc.plot_cost_benefit()
```



The x-axis here is damage averted over the 2018-2080 analysis period. The y-axis is the Benefit/Cost ratio (so higher is better). This means that the area of each shape represents the total benefit of the measure. Furthermore, any measure which goes above 1 on the y-axis gives a larger benefit than the cost of its implementation.

The average annual impact and the total climate risk are marked on the x-axis. The width between the last measure bar and the the total climate risk is the residual risk.

Exercise: How sensitive are cost benefit analyses to different parameters? Let's say an adaptation measure is a 'good investment' if the benefit is greater than the cost over the analysis period, and it's a 'bad investment' if the benefit is less than the cost.

- Using the hazards and exposures from this tutorial, can you design an impact measure that is a good investment when no discount rates are applied, and a bad investment when a 1.4% (or higher) discount rate is applied?

- Create hazard and exposure objects for the same growth and climate change scenarios as this tutorial, but for the year 2040. Can you design an impact measure that is a good investment when evaluated out to 2080, but a bad investment when evaluated to 2040?
- Using the hazards and exposures from this tutorial, can you design an impact measure that is a good investment when `imp_time_depen` = 1/4 (change happens closer to 2018) and a bad investment when `imp_time_depen` = 4 (change happens closer to 2080).

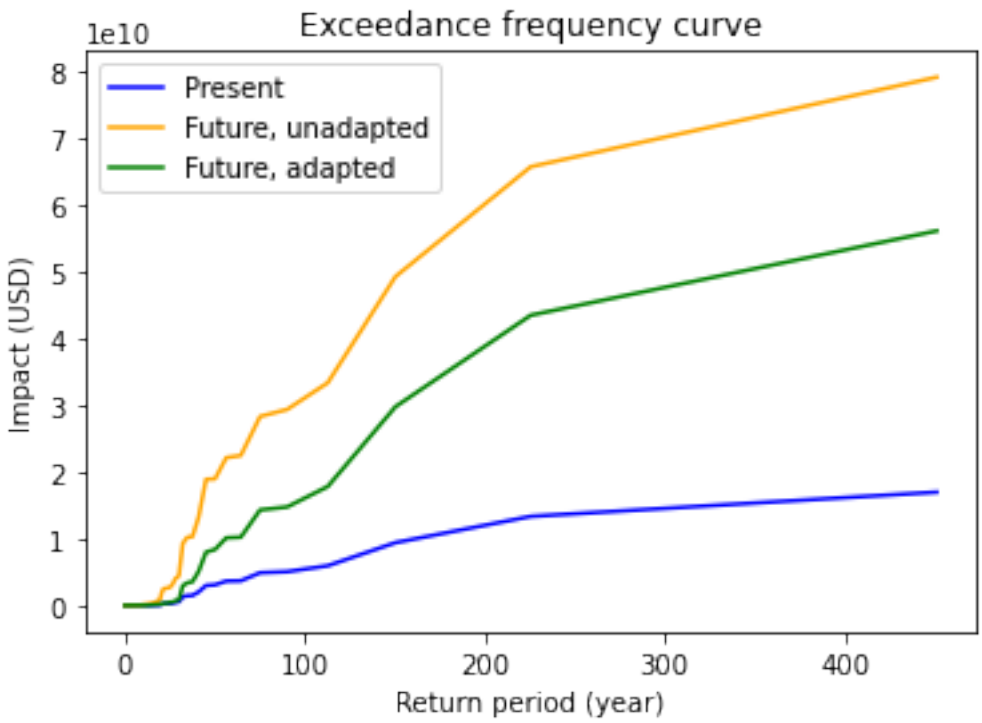
Finally we can use some of the functionality of the objects stored within the CostBenefit object. Remember that many impact calculations have been performed to get here, and if `imp_mat` was set to True, the data has been stored (or … it will be. I found a bug that stops it being saved while writing the tutorial.)

So this means that you can, for example, plot maps of return period hazard with different adaptation measures applied (or with all applied, using `combine_measures`).

Another thing to explore is exceedance curves, which are stored. Here are the curves for the present, future unadapted and future adapted scenarios:

```
combined_costben_disc = costben_disc.combine_measures(
    ["Measure A", "Measure B"],
    "Combined measures",
    new_color=np.array([0.1, 0.8, 0.8]),
    disc_rates=discount_stern,
)
efc_present = costben_disc.imp_meas_present["no measure"]["efc"]
efc_future = costben_disc.imp_meas_future["no measure"]["efc"]
efc_combined_measures = combined_costben_disc.imp_meas_future["Combined measures"][
    "efc"
]

ax = plt.subplot(1, 1, 1)
efc_present.plot(axis=ax, color="blue", label="Present")
efc_future.plot(axis=ax, color="orange", label="Future, unadapted")
efc_combined_measures.plot(axis=ax, color="green", label="Future, adapted")
leg = ax.legend()
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Combined measures             5.22             10.6616         2.04245

--------------------  ---------  --------
Total climate risk:   22.0086    (USD bn)
Average annual risk:   0.984382  (USD bn)
Residual risk:        11.347     (USD bn)
--------------------  ---------  --------
Net Present Values
```



Conclusion

Cost-benefits calculations can be powerful policy tools, but they are as much an art as a science. Describing your adaptation measures well, choosing the period to evaluate them over, describing the changing climate and picking a discount rate will all affect your results and whether a particular measure is worth implementing.

Take the time to explain these choices to yourself and anyone else who wants to understand your calculations. It is also good practice to run sensitivity tests on your results: how much do your conclusions change when you use other plausible setups for the calculation?

2.5.7 Calculate probabilistic impact yearset

This module generates a yearly impact object `yimp` which contains probabilistic annual impacts for a specified amount of years (`sampled_years`). The impact values are extracted from a given impact `imp` object that contains impact values per event. The amount of sampled years as a list of years (`sampled_years`) to be sampled for. The amount of events per sampled year (`events_per_year`) are determined with a Poisson distribution (`lam` = sum(event_impacts.frequency)). Then, the events occurring in each sampled year are sampled from the input `imp` object and summed up per year. Thus, the `yimp` object contains the sum of sampled (event) impacts for each sampled year. In contrast to the expected annual impact (eai), an `yimp` object contains an impact for EACH sampled year and this value differs among years. The number of events_per_year and the selected_events are saved in a sampling vector (`sampling_vect`).

The function `impact_yearset` performs all these computational steps, taking an `imp` and the list of sampled_years (`sampled_years`) as input. The output of the function is the `yimp` object and the `sampling_vect`. Moreover, a `sampling_vect` (generated in a previous run) can be provided as optional input and the user can custom-define the Poisson parameter `lam`. Reapplying the same sampling_vect does not only allow to reproduce the generated `yimp`, but also for a physically consistent way of sampling impacts caused by different hazards.

Sampling options. Per default, impact events are sampled without replacement (given that the original impact object contains enough events). When setting `with_replacement=True`, the impact events are sampled with replacement . Note that sampling without replacement can lead to distorted sampling if the frequencies of the different impacts (`imp.frequency`) are not equal.

Correction factor. By default, a correction factor is applied uniformly to all yearly impacts, such that the final `yimp` object has the same average annual impact than `imp`, the original impact object. Applying the correction factor can be avoided by setting `correction_fac=False`.

To make the process more transparent, this tutorial shows the single computations that are performed when generating an `yimp` object for a dummy event_impacts object.

```
import numpy as np
import climada.util.yearsets as yearsets
from climada.engine import Impact

# dummy event_impacts object containing 12 event impacts
imp = Impact(
    event_id=np.arange(6) + 10,
    event_name=np.arange(6) + 10,
    date=np.arange(6),
    coord_exp=np.array([[1, 2], [1.5, 2.5]]),
    eai_exp=np.array([13.4, 13.4]),
    at_event=np.array([0, 2, 4, 6, 60, 62]),
    frequency=np.full(6, 0.2),
    aai_agg=26.8,
)
```

Step-by-step calculation

```
# the number of years to sample impacts for (length(yimp.at_event) = n_sampled_years)
n_sampled_years = 10

# sample number of events per sampled year
lam = np.sum(imp.frequency)
events_per_year = yearsets.sample_from_poisson(n_sampled_years, lam)
events_per_year
```

```
array([0, 1, 1, 0, 2, 3, 3, 0, 2, 3])
```

```
# generate the sampling vector
sampling_vect = yearsets.sample_events(events_per_year, imp.frequency)
sampling_vect
```

```
[array([], dtype=int64),
 array([0]),
 array([1]),
 array([], dtype=int64),
 array([4, 2]),
 array([2, 0, 4]),
 array([5, 1, 3]),
 array([], dtype=int64),
 array([5, 3]),
 array([5, 3, 1])]
```

```
# calculate the impact per year
imp_per_year = yearsets.compute_imp_per_year(imp, sampling_vect)
imp_per_year
```

```
array([ 0,  0,  2,  0, 64, 64, 70,  0, 68, 70])
```

```
# calculate the correction factor
correction_factor = yearsets.calculate_correction_fac(imp_per_year, imp)
correction_factor
```

```
0.7928994082840237
```

Direct calculation

If we use the same seed, the `yimp.at_event` values coincide with the step-by-step `imp_per_year` values, if no correction factor is applied.

```
example_seed = 12345  # fix a seed
sampled_years = list(range(2015, 2025))

# direct computation
yimp, sampling_vect = yearsets.impact_yearset(
    imp, sampled_years, correction_fac=False, seed=example_seed
)

# step-by-step computation without applying the correction factor
events_per_year = yearsets.sample_from_poisson(
    len(sampled_years), lam, seed=example_seed
)
sampling_vect = yearsets.sample_events(
    events_per_year, imp.frequency, seed=example_seed
)
imp_per_year = yearsets.compute_imp_per_year(imp, sampling_vect)


print("yimp.at_event = ", yimp.at_event)
print("imp_per_year = ", imp_per_year)
print(
    f"The expected annual impact {round(yimp.aai_agg,2)} differs from the one of the␣
↪original impact ({round(imp.aai_agg,2)})."
)
```

```
yimp.at_event =  [  2   0   0  70 130   0   0 128  68  62]
imp_per_year =  [  2   0   0  70 130   0   0 128  68  62]
The expected annual impact 46.0 differs from the one of the original impact (26.8).
```

Similarly, if we use the same seed, the `yimp.at_event` values when using the correction factor coincide with the step-by-step `imp_per_year` values after applyng the correction factor.

```
# direct computation
yimp, sampling_vect = yearsets.impact_yearset(
    imp, sampled_years, correction_fac=True, seed=example_seed
)

# compute correction factor from imp_per_year computed above without correction factor
correction_factor = yearsets.calculate_correction_fac(imp_per_year, imp)
```

(continues on next page)

(continued from previous page)

```
print("yimp.at_event = ", np.round(yimp.at_event, 2))
print("imp_per_year = ", np.round(imp_per_year * correction_factor, 2))
print(
    f"The expected annual impact {round(yimp.aai_agg,2)} is equal to the one of the␣
↪original impact ({round(imp.aai_agg,2)})."
)
```

```
2025-09-19 15:39:54,743 - climada.util.yearsets - INFO - The correction factor is 0.
↪5826086956521739.
yimp.at_event =  [ 1.17  0.    0.   40.78 75.74  0.    0.   74.57 39.62 36.12]
imp_per_year =  [ 1.17  0.    0.   40.78 75.74  0.    0.   74.57 39.62 36.12]
The expected annual impact 26.8 is equal to the one of the original impact (26.8).
```

2.6 Local exceedance intensities, local exceedance impacts, and return periods

This tutorial presents methods available for `Hazard` and `Impact` objects, to compute local exceedance values and local return periods. In particular, the available methods compute local exceedance intensities (`Hazard.local_exceedance_intensity`) and local exceedance impacts (`Impact.local_exceedance_impact`) for user-defined return periods, and local return periods for user-defined threshold values (`Hazard.local_return_period` or `Impact.local_return_period`).

We first explain the methods functionality and options using a mock Hazard object such that the computation can be easily followed. Further below, we apply the methods to realistic Hazard and Impact objects. If you are already familiar with local exceedance values and return values, you can directly jump to the section about *Method comparison for a realistic Hazard object*.

2.6.1 Demonstration using a mock Hazard object

Define a mock Hazard object

We define a simple mock TC Hazard object with which we will demonstrate the methods and their different parameter choices. The Hazard object consists of two events and has a spatial extent of four centroids, A, B, C and D. The first event has a (estimated) frequency of 9 times every 100years, and the second even has a (estimated) frequency of once every 100years. The two events have the following spatial intensity pattern (in unit m/s)

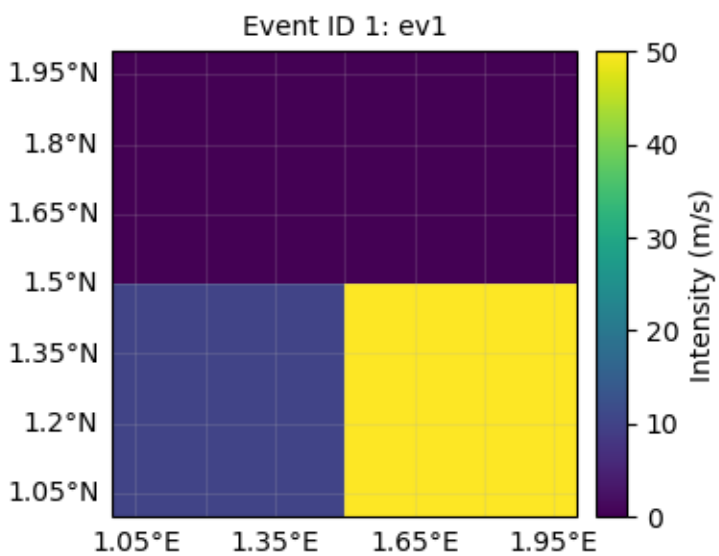
event_id	centroid A	centroid B	centroid C	centroid D
1	0	0	10	50
2	0	100	100	100

```
# import packages
import numpy as np
from scipy import sparse
import warnings

warnings.filterwarnings("ignore")
```

(continues on next page)

(continued from previous page)

```
from climada.hazard.base import Hazard
from climada.hazard.centroids.centr import Centroids
from climada.util.plot import plot_from_gdf

# hazard intensity
intensity = sparse.csr_matrix([[0, 0, 10, 50], [0, 100, 100, 100]])

# hazard centroids
centroids = Centroids(lat=np.array([2, 2, 1, 1]), lon=np.array([1, 2, 1, 2]))

# define hazard
hazard = Hazard(
    haz_type="TC",
    intensity=intensity,
    fraction=np.full_like(intensity, 1),
    centroids=centroids,
    event_id=np.array([1, 2]),
    event_name=["ev1", "ev2"],
    date=np.array([1, 2]),
    orig=np.array([True, True]),
    frequency=np.array([9.0 / 100, 1.0 / 100]),
    frequency_unit="1/year",
    units="m/s",
)

# plot first event of Hazard object
hazard.plot_intensity(event=1, smooth=False, figsize=(4, 4));
```



Compute local exceedance intensities

Given the information of the Hazard object, which hazard intensity do we expect at each centroid to reoccur at a given return period? For instance, we could ask which intensity to expect every 5, 30, and 150 years. This question is addressed by the method `Hazard.local_exceedance_intensity()` which we will explain in the following, including different parameter settings one can choose from.

To compute which intensity to expect at a centroid for a given return period, we have to infer from the Hazard object how often different intensities are exceeded. To do so, we sort the events according to their intensity at the centroid, and then, for each intensity, sum the frequencies of the events that exceed this intensity. This resulting cumulative frequency for each intensity then yields the intensity's return period as one over the cumulative frequency. Finally, as we see below, the return periods for new intensities have to be inter- and extrapolated from the Hazard objects data.

Examplary calculation for centroid D

We demonstrate this computation for centroid D. First, we sort the centroid's intensities in descending order. Centroid D has the two intensities `intensities_descending = [100, 50]`. Then, the corresponding frequencies will be accumulated in this order, giving the cumulative frequencies `cumulative_frequencies = [0.01, 0.1]` (note that 9/100 + 1/100 = 1/10). The return periods of the different intensities are then given by `1/cummulative_frequencies`. This means that our data for centroid D suggests that, e.g., we expect an event with an intensity that exceeds 100m/s on average every 100 years. The information for centroid D is summarized as

intensity (m/s)	return period (years)
100	100
50	10

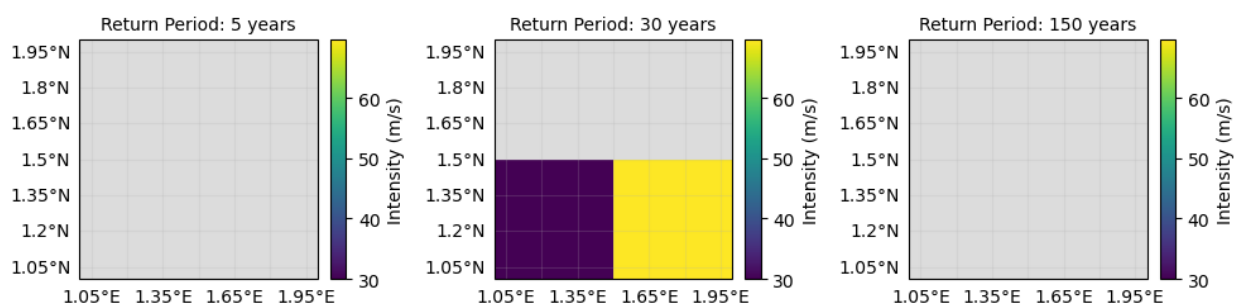
Now, the question is how to estimate the exceedance intensity for new return periods, e.g., 5, 30, and 150 years.

Option 1 (default setting): interpolation without extrapolation

For the return periods inside the range of observed return periods (here, 30 years), one can simply interpolate between the data points. For return periods outside the range of observed return periods (here, 5 and 150 years), the cautious answer is "we don't know" and one returns NaN. This behaviour is given using `method='interpolate'` which is the default setting.

Note that, by default, the linear interpolation between data points is done after converting the data to logarithmic scale. We do this because, when extrapolating, logarithmic scales avoid negative numbers. The scale choice can be controlled by changing the boolean parameters `log_frequency` and `log_intensity`.

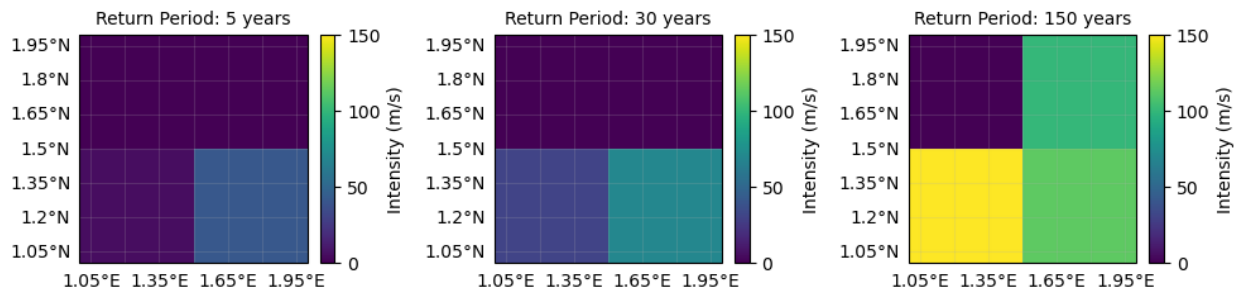
```
local_exceedance_intensity, title, column_label = hazard.local_exceedance_intensity(
    return_periods=[5, 30, 150], method="interpolate"
)
plot_from_gdf(
    local_exceedance_intensity, title, column_label, smooth=False, figsize=(10, 6)
);
```



Option 2: extrapolation

If the user wants to estimate the return periods outside the range of observed return periods (here, 5 and 150 years), they can use `method='extrapolate'`. This just extends the last interpolation piece inside the data range beyond the data borders. If there is only a single (nonzero) data point, this setting returns the given intensity (e.g., 100m/s for centroid B) for return periods above the observed return period (e.g., 100 years for centroid B), and zero intensity for return periods below. Centroids where all events have zero intensity will be assigned zero exceedance intensity for any return period.

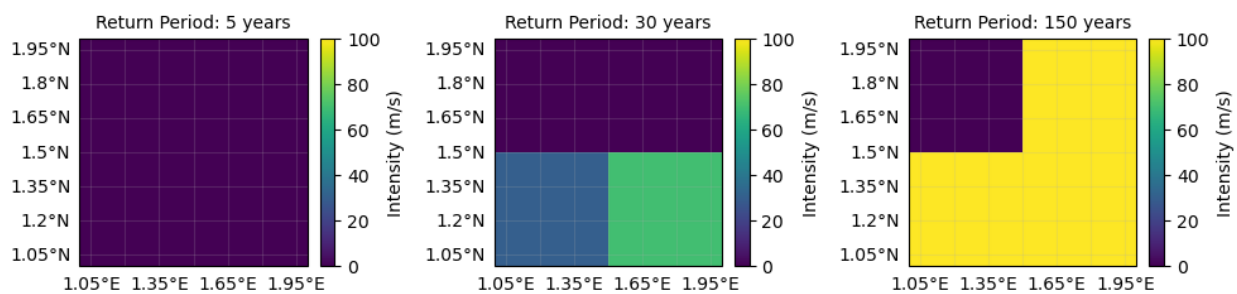
```
local_exceedance_intensity, title, column_label = hazard.local_exceedance_intensity(
    return_periods=[5, 30, 150], method="extrapolate"
)
plot_from_gdf(
    local_exceedance_intensity, title, column_label, smooth=False, figsize=(10, 6)
);
```



Option 3: extrapolation with constant values

Users who want to extrapolate in a more cautious way can use `method='extrapolate_constant'`. Here, return periods above the largest obsvered return period are assigned the largest intensity, and return periods below the smallest observed return periods are assigned 0.

```
local_exceedance_intensity, title, column_label = hazard.local_exceedance_intensity(
    return_periods=[5, 30, 150], method="extrapolate_constant"
)
plot_from_gdf(
    local_exceedance_intensity, title, column_label, smooth=False, figsize=(10, 6)
);
```

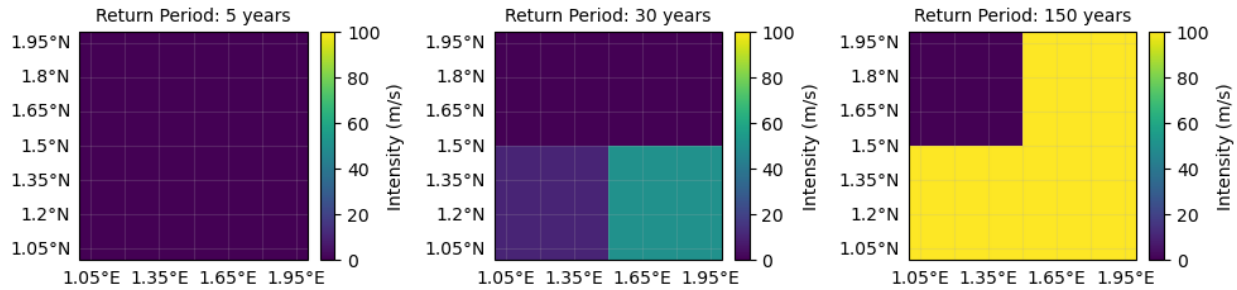


Option 4: stepfunction

Finally, instead of interpolating between the data points, one can use `method='stepfunction'`. Here, a user-provided return period will be assigned an exceedance intensity equal to the intensity corresponding to the closest observed return period that is below the given the user-provided return period. The extrapolation behaviour is the same as for

method='extrapolate_constant'.

```
local_exceedance_intensity, title, column_label = hazard.local_exceedance_intensity(
    return_periods=[5, 30, 150], method="stepfunction"
)
plot_from_gdf(
    local_exceedance_intensity, title, column_label, smooth=False, figsize=(10, 6)
);
```

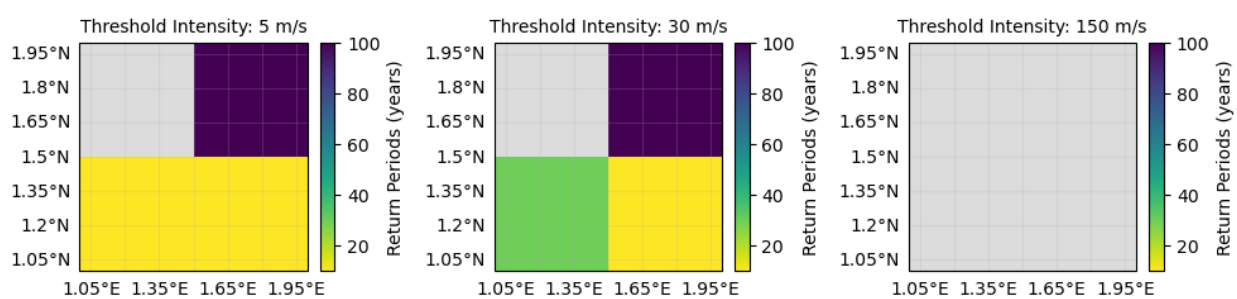


Compute local return periods

Using `Hazard.local_return_period()`, you can locally compute the return period of events exceeding different (user-specified) threshold intensities. For instance, you could ask every how many years (on average) do we expect intensities exceeding 5m/s, 30m/s, and 150m/s.

The different settings of `Hazard.local_return_period()` are similar to the ones available in `Hazard.local_exceedance_intensity()`, see Options 1-4 above. For instance, if we use `method='extrapolate_constant'`, we obtain the local return periods shown below. Note that the main difference to the local exceedance intensities described above is that, for a threshold intensity above all observed intensities, the method assigns NaN, and for a threshold intensity below all observed intensities, the method assigns the smallest observed return period.

```
# method: extrapolation
local_return_period, title, column_label = hazard.local_return_period(
    threshold_intensities=[5, 30, 150], method="extrapolate_constant"
)
plot_from_gdf(local_return_period, title, column_label, smooth=False, figsize=(10,␣
↪6));
```

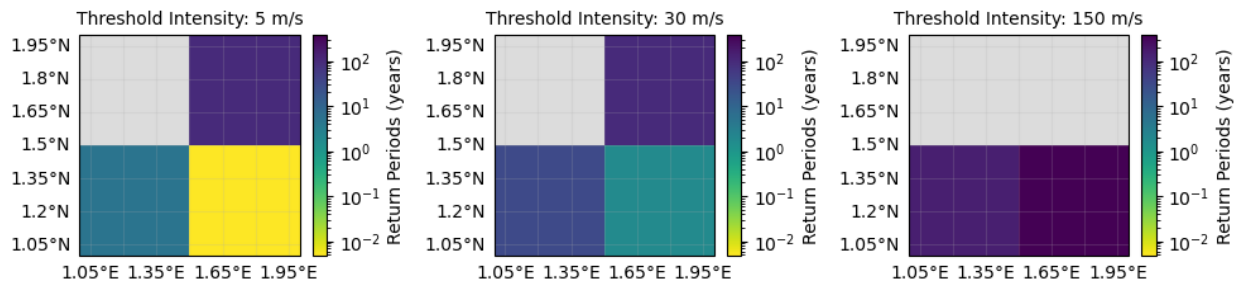



```
# method: extrapolation
local_return_period, title, column_label = hazard.local_return_period(
    threshold_intensities=[5, 30, 150], method="extrapolate"
)
```

(continues on next page)

(continued from previous page)

```
plot_from_gdf(local_return_period, title, column_label, smooth=False, figsize=(10,␣
↪6));
```



2.6.2 Method comparison for a realistic Hazard object

We now showcase the different settings for `Hazard.local_exceedance_intensity()` with a real example (historic tropical cyclones in Florida from 1990 to 2004). First, we read in the hazard object.

```
# load example hazard object
from climada.hazard import Hazard, Centroids
from climada.test import get_test_file

haz_tc_fl = Hazard.from_hdf5(
    get_test_file("HAZ_DEMO_FL_15")
)  # Historic tropical cyclones in Florida from 1990 to 2004
haz_tc_fl.check()  # Use always the check() method to see if the hazard has been␣
↪loaded correctly
```

```
2025-03-31 17:21:06,957 - climada.hazard.io - INFO - Reading /Users/vgebhart/climada/
↪data/hazard/template/HAZ_DEMO_FL_15/v1/HAZ_DEMO_FL_15.h5
```

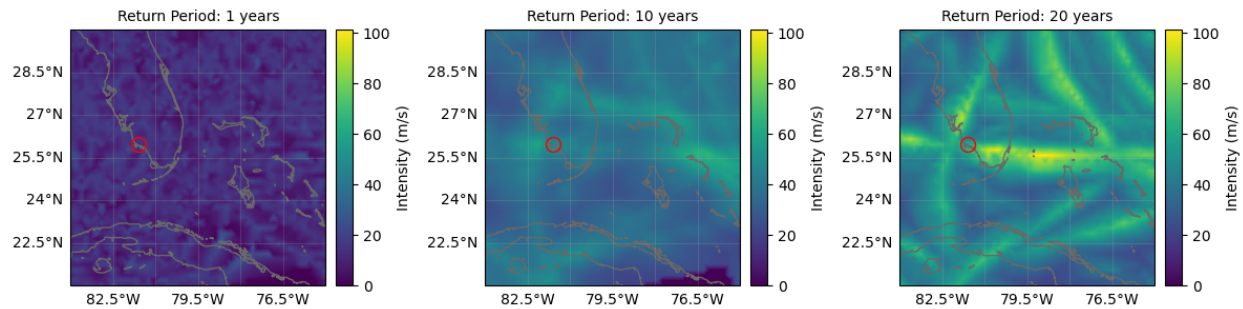
Next, we plot the hazards local exceedance intensities for user-specified return periods of 10, 100 and 200 years. We use the setting `"method=extrapolate"`. Furthermore, we indicate a specific centroid (red circle) that we will analyse in more detail below.

```
import cartopy.crs as ccrs

# choose centroid index for analysis below
i_centroid = 1363
coordinates_centroid = haz_tc_fl.centroids.coord[i_centroid]

local_exceedance_intensity, title, column_label = haz_tc_fl.local_exceedance_
↪intensity(
    return_periods=[1, 10, 20], method="extrapolate"
)
axes = plot_from_gdf(local_exceedance_intensity, title, column_label, figsize=(12, 8))
for axis in axes:
    axis.plot(
        coordinates_centroid[1],
        coordinates_centroid[0],
        marker="o",
        markersize=10,
```

(continues on next page)

(continued from previous page)

```
        markerfacecolor="none",
        markeredgecolor="red",
        transform=ccrs.PlateCarree(),
    )
```



Now, we calculate the local exceedance frequencies for return periods ranging from 5 to 250 years, using the four different options explained above.

```
test_return_periods = np.arange(0.5, 25, 0.1)
interpolated = haz_tc_fl.local_exceedance_intensity(return_periods=test_return_
↪periods)[
    0
]
extrapolated = haz_tc_fl.local_exceedance_intensity(
    return_periods=test_return_periods, method="extrapolate"
)[0]
extrapolated_constant = haz_tc_fl.local_exceedance_intensity(
    return_periods=test_return_periods, method="extrapolate_constant"
)[0]
stepfunction = haz_tc_fl.local_exceedance_intensity(
    return_periods=test_return_periods, method="stepfunction"
)[0]
```

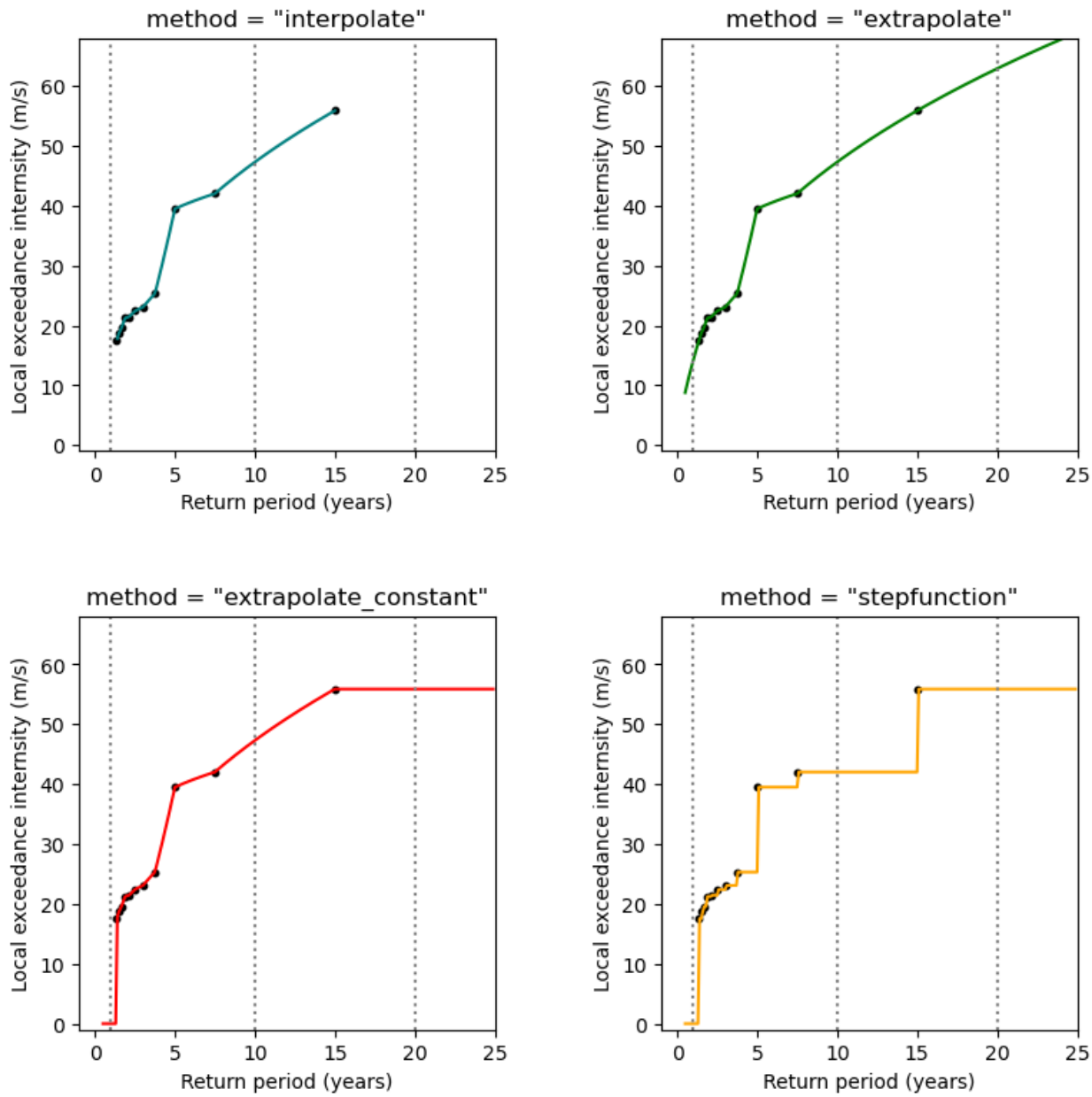
Finally, we focus on a specific centroid (red circle in above plots) and show how the different options of `Hazard.local_exceedance_intensity()` can lead to different results. The user-specified return periods from above are indicated as dotted lines. Note in particular that the return periods 5 years and 200 years that we considered above, lie outside the range of observed values for this centroid (blue scatter points). Thus, depending on the extrapolation choice, `Hazard.local_exceedance_intensity()` either returns NaN (`"method=interpolate"`, default option) or different extrapolated estimates.

```
# plot different extrapolation methods at at centroid
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2, figsize=(9, 9))
plt.subplots_adjust(wspace=0.4, hspace=0.4)

for axis, local_exceedance_intensity, color, title in zip(
    axes.flatten(),
    [interpolated, extrapolated, extrapolated_constant, stepfunction],
    ["teal", "g", "r", "orange"],
    ["interpolate", "extrapolate", "extrapolate_constant", "stepfunction"],
):
```

(continues on next page)

(continued from previous page)

```
    axis.plot(
        test_return_periods,
        local_exceedance_intensity.values[i_centroid, 1:],
        color=color,
    )
    axis.set_ylabel("Local exceedance internsity (m/s)")
    axis.set_xlabel("Return period (years)")
    axis.scatter(
        1
        / (
            haz_tc_fl.frequency[0] * np.arange(11, 0, -1)
        ),  # sorted return periods of intensity values at centroid
        np.sort(np.unique(haz_tc_fl.intensity[:, i_centroid].toarray()))[
            1:
        ],  # sorted intensity values at centroid
        s=10,
        color="k",
    )
    axis.vlines([1, 10, 20], ymin=-1, ymax=70, linestyles=":", colors="gray")
    axis.set_title(f'method = "{title}"')
    axis.set_xlim([-1, 25])
    axis.set_ylim([-1, 68])
```



2.6.3 Compute local exceedance impacts and local return periods of impact objects

Completely analogous to the above explained methods `Hazard.local_exceedance_intensity()` and `Hazard.local_return_period()` of a `Hazard` object, an `Impact` object has the methods `Impact.local_exceedance_impact()` and `Impact.local_return_period()` (to be added soon).

```
# prepare hazard object

import warnings

warnings.filterwarnings("ignore")
```

(continues on next page)

(continued from previous page)

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone
from climada.engine import ImpactCalc
from climada.util.api_client import Client
from climada.util.constants import CMAP_IMPACT
from climada.util.plot import plot_from_gdf

client = Client()
haz_tc_haiti = client.get_hazard(
    "tropical_cyclone",
    properties={
        "country_name": "Haiti",
        "climate_scenario": "historical",
        "nb_synth_tracks": "10",
    },
)
haz_tc_haiti.check()
```

```
2025-03-31 17:21:31,356 - climada.hazard.io - INFO - Reading /Users/vgebhart/climada/
↪data/hazard/tropical_cyclone/tropical_cyclone_10synth_tracks_150arcsec_HTI_1980_
↪2020/v2/tropical_cyclone_10synth_tracks_150arcsec_HTI_1980_2020.hdf5
```

```
# prepare exposure

exposure = client.get_litpop(country="Haiti")
exposure.check()
```

```
2025-03-31 17:21:32,751 - climada.entity.exposures.base - INFO - Reading /Users/
↪vgebhart/climada/data/exposures/litpop/LitPop_150arcsec_HTI/v3/LitPop_150arcsec_HTI.
↪hdf5
2025-03-31 17:21:32,768 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2025-03-31 17:21:32,769 - climada.entity.exposures.base - INFO - category_id not set.
2025-03-31 17:21:32,769 - climada.entity.exposures.base - INFO - cover not set.
2025-03-31 17:21:32,769 - climada.entity.exposures.base - INFO - deductible not set.
2025-03-31 17:21:32,770 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
# prepare impact function

impf_tc = ImpfTropCyclone.from_emanuel_usa()
impf_set = ImpactFuncSet([impf_tc])
impf_set.check()
```

```
# compute impact

impact = ImpactCalc(exposure, impf_set, haz_tc_haiti).impact(save_mat=True)
```

```
2025-03-31 17:21:32,824 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2025-03-31 17:21:32,825 - climada.entity.exposures.base - INFO - Matching 1329␣
↪exposures with 1332 centroids.
```

(continues on next page)

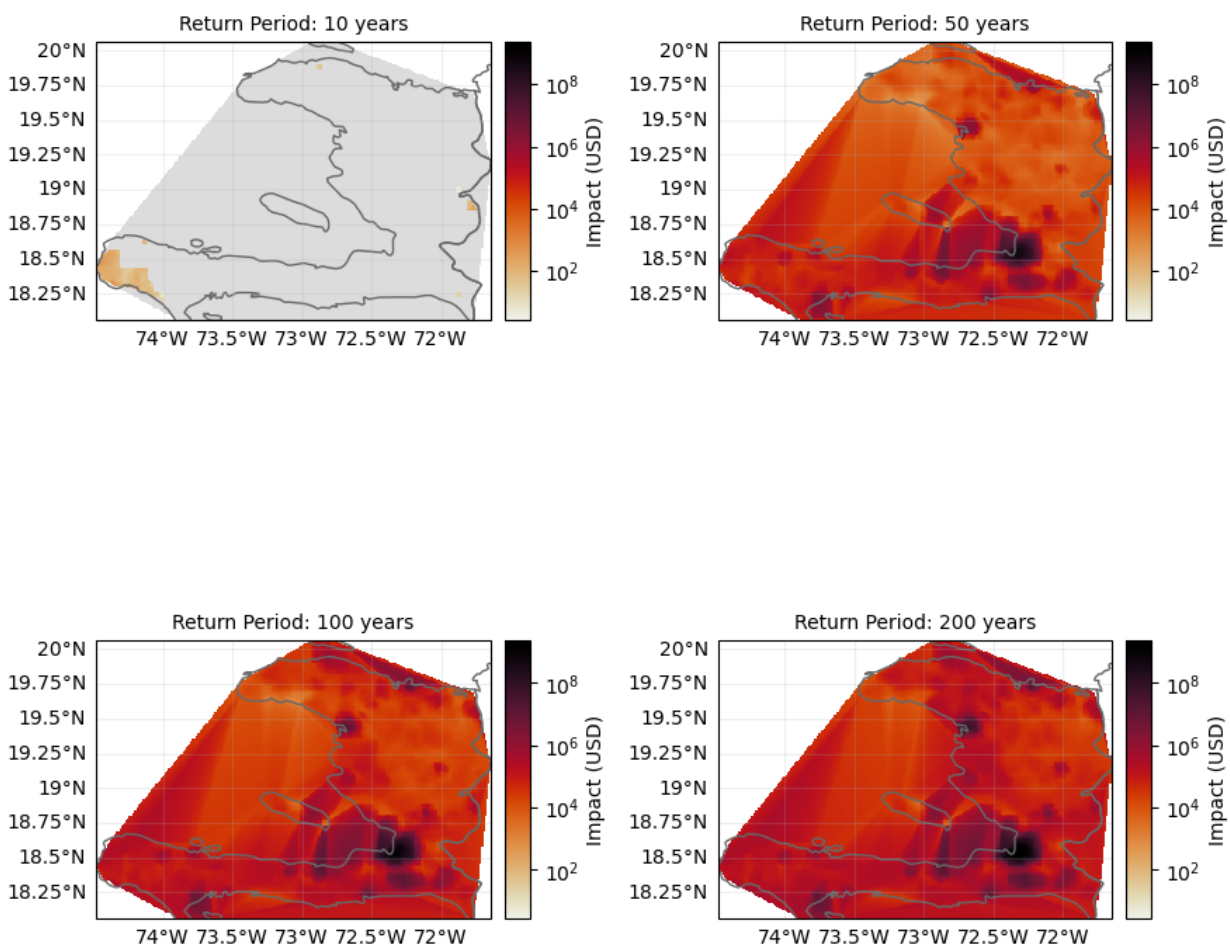
(continued from previous page)

```
2025-03-31 17:21:32,827 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2025-03-31 17:21:32,830 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪3987 assets (>0) and 43560 events.
```

```
# compute local exceedance impacts

return_periods = [10, 50, 100, 200]
loacl_exceedance_impacts, title, column_label = impact.local_exceedance_impact(
    return_periods=return_periods
)
```

```
2025-03-31 17:21:32,861 - climada.engine.impact - INFO - Computing exceedance impact␣
↪map for return periods: [10, 50, 100, 200]
```

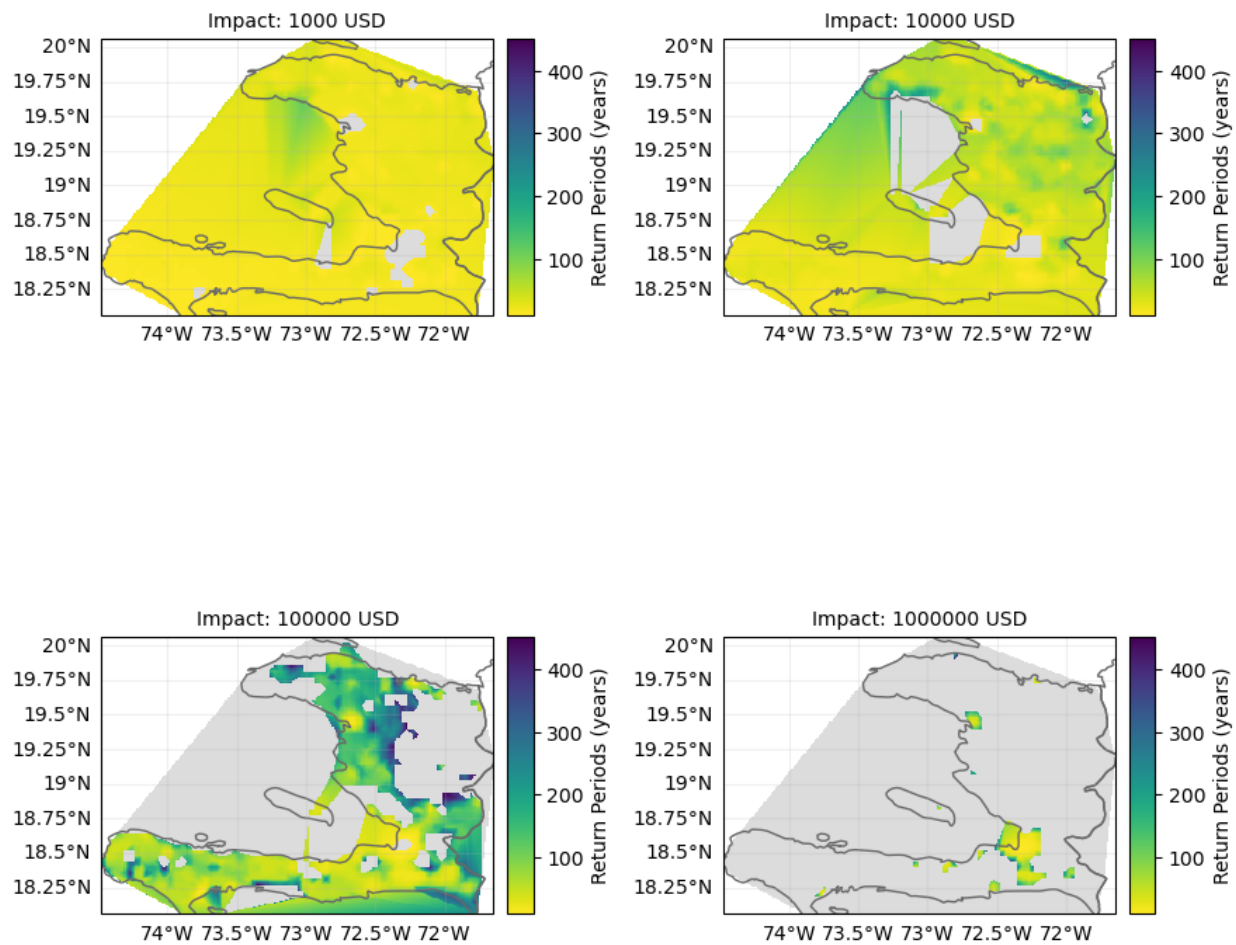
```
# plot local exceedance impacts
plot_from_gdf(
    loacl_exceedance_impacts, title, column_label, smooth=False, cmap=CMAP_IMPACT
);
```



```
# compute local return periods of impact
threshold_impact = [1000, 10000, 100000, 1000000]
local_return_periods, title, column_label = impact.local_return_period(
    threshold_impact=threshold_impact
)
```

```
2025-03-31 17:21:58,100 - climada.engine.impact - INFO - Computing return period map␣
↪for impacts: [1000, 10000, 100000, 1000000]
```

```
# plot local return periods of impacts
plot_from_gdf(local_return_periods, title, column_label, smooth=False);
```



2.6.4 When to use binning in calculating local exceedance frequencies or local return periods

When using `method='extrapolate'`, local exceedance frequencies or local return periods are extrapolated beyond the observed data range. The used extrapolation technique is the one of `scipy.interpolate.interp1d`: the two extremal interpolations at the lower and upper limits of the data are simply extended beyond the data range.

In specific cases (see below for an example), this can lead to undesired behaviour. For this reason, the user can pass in an additional integer parameter `bin_decimals` to bin the intensities (and sum up the corresponding frequencies) according to `bin_decimals` decimal places.

As can be seen in the following example, this binning can lead to a different and more stable extrapolation (which might be desriable in particular for the `local_return_period` method), and to a smoother interpolation. Note that, due to binning, the data range of observed return periods for `method='interpolate'` may be reduced.

As an example, we consider a hazard object that contains one centroid and five intensities, all of which occur with a frequency of 1/10years. Importantly, the two maximal intensities are very similar, which strongly affects the extrapolation behaviour.

```
hazard_binning = Hazard(
    intensity=sparse.csr_matrix([[80.0], [80.02], [70.0], [70.0], [60.0]]),
    frequency=np.array([0.1, 0.1, 0.1, 0.1, 0.1]),
    frequency_unit="1/year",
    units="m/s",
    centroids=Centroids(lat=np.array([1]), lon=np.array([2])),
)
```

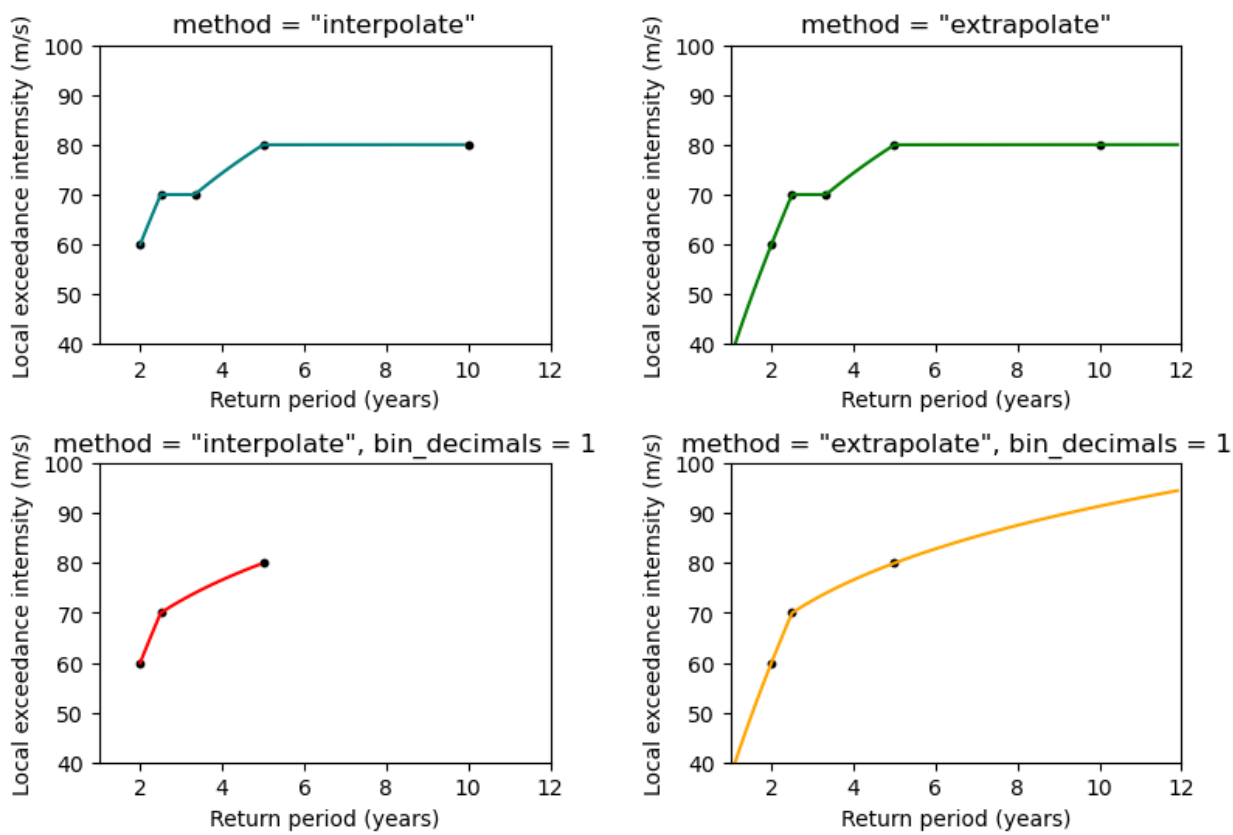
```
test_return_periods = np.arange(1, 12, 0.1)
interpolated = hazard_binning.local_exceedance_intensity(
    return_periods=test_return_periods
)[0]
extrapolated = hazard_binning.local_exceedance_intensity(
    return_periods=test_return_periods, method="extrapolate"
)[0]
interpolated_binned = hazard_binning.local_exceedance_intensity(
    return_periods=test_return_periods, bin_decimals=1
)[0]
extrapolated_binned = hazard_binning.local_exceedance_intensity(
    return_periods=test_return_periods, method="extrapolate", bin_decimals=1
)[0]
```

```
# plot different extrapolation methods at at centroid
import matplotlib.pyplot as plt

fig, axes = plt.subplots(2, 2, figsize=(9, 6))
plt.subplots_adjust(wspace=0.4, hspace=0.4)

for axis, local_exceedance_intensity, color, title in zip(
    axes.flatten(),
    [interpolated, extrapolated, interpolated_binned, extrapolated_binned],
    ["teal", "g", "r", "orange"],
    [
        '"interpolate"',
        '"extrapolate"',
        '"interpolate", bin_decimals = 1',
        '"extrapolate", bin_decimals = 1',
    ],
):
    axis.plot(
        test_return_periods,
        local_exceedance_intensity.values[0, 1:],
        color=color,
    )
    axis.set_ylabel("Local exceedance internsity (m/s)")
```

(continues on next page)

(continued from previous page)

```
    axis.set_xlabel("Return period (years)")
    if title in ['"interpolate"', '"extrapolate"']:
        cum_freq = np.arange(5, 0, -1) * hazard_binning.frequency[0]
        intensities = np.sort(hazard_binning.intensity[:, 0].toarray())[::-1]
    else:
        cum_freq = np.array([5, 4, 2]) * hazard_binning.frequency[0]
        intensities = np.sort(
            np.unique(np.round(hazard_binning.intensity[:, 0], decimals=1).toarray())
        )
    axis.scatter(
        1 / cum_freq,
        intensities,
        s=10,
        color="k",
    )
    axis.set_title(f"method = {title}")
    axis.set_xlim([1, 12])
    axis.set_ylim([40, 100])
```



Given the hazard distribution in the hazard object (black points), the user might prefer flat extrapolation behaviour to large return periods, in which case the default `bin_decimals=None` is the correct choice.

However, in the inverse problem of computing a return period for a given hazard intensity, not binning the values can lead to an unbounded extrapolation, and binning the values might be a good choice.

```
test_hazard_intensities = [55.0, 75.0, 85.0]
print("Local return periods [years] without binning:\n")
print(
    hazard_binning.local_return_period(
        threshold_intensities=test_hazard_intensities,
        method="extrapolate",
    )[0].to_markdown()
)
print("\n\nLocal return periods [years] with binning to bin_decimals=1:\n")
print(
    hazard_binning.local_return_period(
        threshold_intensities=test_hazard_intensities,
        method="extrapolate",
        bin_decimals=1,
    )[0].to_markdown()
)
```

```
Local return periods [years] without binning:

|    | geometry    |    55.0 |    75.0 |        85.0 |
|---:|:------------|--------:|--------:|------------:|
|  0 | POINT (2 1) | 1.76331 | 4.11019 | 5.09816e+73 |


Local return periods [years] with binning to bin_decimals=1:

|    | geometry    |    55.0 |    75.0 |    85.0 |
|---:|:------------|--------:|--------:|--------:|
|  0 | POINT (2 1) | 1.76331 | 3.57665 | 6.84921 |
```

2.7 Uncertainty Quantification Tutorials

2.7.1 Unsequa - a module for uncertainty and sensitivity analysis

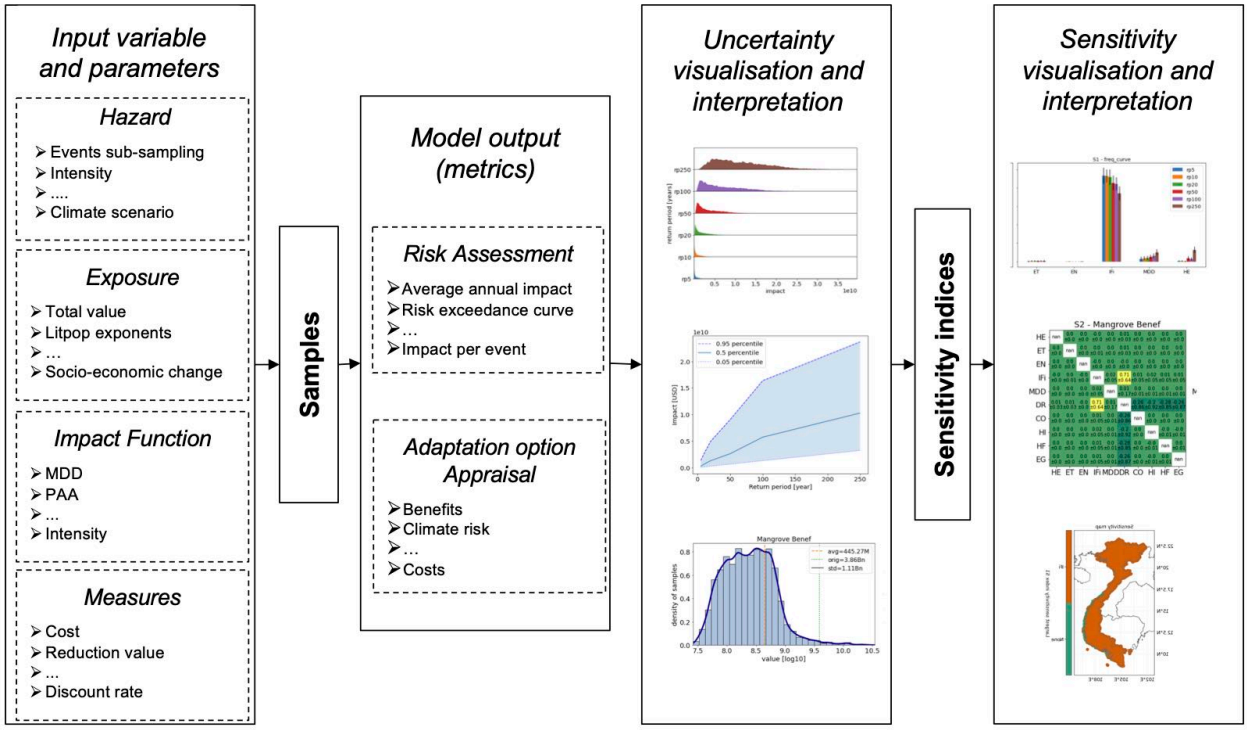
This is a tutorial for the unsequa module in CLIMADA. A detailled description can be found in Kropf (2021).

Uncertainty and sensitivity analysis

Before doing an uncertainty quantification in CLIMADA, it is imperative that you get first comfortable with the different notions of uncertainty in the modelling world (see e.g.Pianosi (2016) or Douglas-Smith(2020) for a review). In particular, note that the uncertaintity values will only be as good as the input from the user. In addition, not all uncertainties can be numerically quantified, and even worse, some unkonwns are unknown. This means that sometimes, quantifying uncertainty can lead to false confidence in the output!. For a more philosophical discussion about the types of uncertainties in climate research see Knüsel (2020) and Otth (2022).

In this module, it is possible to perform global uncertainty analysis, as well as a sensitivity analysis. The word global is meant as opposition to the 'one-factor-at-a-time' (OAT) strategy. The OAT strategy, which consists in analyzing the effect of varying one model input factor at a time while keeping all other fixed, is popular among modellers, but has major shortcomings Saltelli (2010), Saltelli(2019) and should not be used.

A rough schemata of how to perform uncertainty and sensitivity analysis (taken from Kropf(2021))



1. Kropf, C.M. et al. Uncertainty and sensitivity analysis for global probabilistic weather and climate risk modelling: an implementation in the CLIMADA platform (2021)
2. Pianosi, F. et al. Sensitivity analysis of environmental models: A systematic review with practical workflow. Environmental Modelling & Software 79, 214–232 (2016). 3.Douglas-Smith, D., Iwanaga, T., Croke, B. F. W. & Jakeman, A. J. Certain trends in uncertainty and sensitivity analysis: An overview of software tools and techniques. Environmental Modelling & Software 124, 104588 (2020)
3. Knüsel, B. Epistemological Issues in Data-Driven Modeling in Climate Research. (ETH Zurich, 2020)
4. Saltelli, A. et al. Why so many published sensitivity analyses are false: A systematic review of sensitivity analysis practices. Environmental Modelling & Software 114, 29–39 (2019)
5. Saltelli, A. & Annoni, P. How to avoid a perfunctory sensitivity analysis. Environmental Modelling & Software 25, 1508–1517 (2010)

Unsequa Module Structure

The unsequa module contains several key classes.

The model input parameters and their distribution are specified as

- `InputVar`: defines input uncertainty variables

The input parameter sampling, Monte-Carlo uncertainty distribution calculation and the sensitivity index computation are done in

- `CalcImpact`: compute uncertainties for outputs of `climada.engine.impact.calc` (child class of `Calc`)
- `CalcDeltaImpact`: compute uncertainties for outputs of `climada.engine.impact.calc` (child class of `Calc`)

- `CalcCostBenefit`: compute uncertainties for outputs of `climada.engine.cost_benefit.calc` (child class of `Calc`)

The results are stored in

- `UncOutput`: store the uncertainty and sensitivity analysis results. Contains also several plotting methods. This is a class which only stores data.
- `UncImpactOutput`: subclass with dataframes specifically for `climada.engine.impact.calc` uncertainty and sensitivity analysis results.
- `UncCostBenefitOutput`: subclass with dataframes specifically for `climada.engine.cost_benefit.calc` uncertainty and sensitivity analysis results.

InputVar

The InputVar class is used to define uncertainty variables.

Attribute	Type	Description
func	function	Model variable defined as a function of the uncertainty input parameters
distr_dict	dict	Dictionary of the probability density distributions of the uncertainty input parameters

An **input uncertainty parameter** is a numerical input value that has a certain probability density distribution in your model, such as the total exposure asset value, the slope of the vulnerability function, the exponents of the litpop exposure, the value of the discount rate, the cost of an adaptation measure, …

The probability densitity distributions (values of `distr_dict`) of the input uncertainty parameters (keyword arguments of the `func` and keys of the `distr_dict`) can be any of the ones defined in scipy.stats.

Several helper methods exist to make generic InputVar for Exposures, ImpactFuncSet, Hazard, Entity (including DiscRates and Measures). These are described in details in the tutorial *Helper methods for InputVar*. These are a good bases for your own computations.

Example - custom continuous uncertainty parameter

Suppose we assume that the GDP value used to scale the exposure has a relative error of +-10%.

```
import warnings

warnings.filterwarnings("ignore")  # Ignore warnings for making the tutorial's pdf.

# Define the base exposure
from climada.util.constants import EXP_DEMO_H5
from climada.entity import Exposures

exp_base = Exposures.from_hdf5(EXP_DEMO_H5)
```

```
# Define the function that returns an exposure with scaled total assed value
# Here x_exp is the input uncertainty parameter and exp_func the inputvar.func.
def exp_func(x_exp, exp_base=exp_base):
    exp = exp_base.copy()
```

(continues on next page)

(continued from previous page)

```
    exp.gdf["value"] *= x_exp
    return exp
```

```
# Define the Uncertainty Variable with +-10% total asset value
# The probability density distribution of the input uncertainty parameter x_exp is sp.
↪stats.uniform(0.9, 0.2)
from climada.engine.unsequa import InputVar
import scipy as sp

exp_distr = {
    "x_exp": sp.stats.uniform(0.9, 0.2),
}
exp_iv = InputVar(exp_func, exp_distr)
```

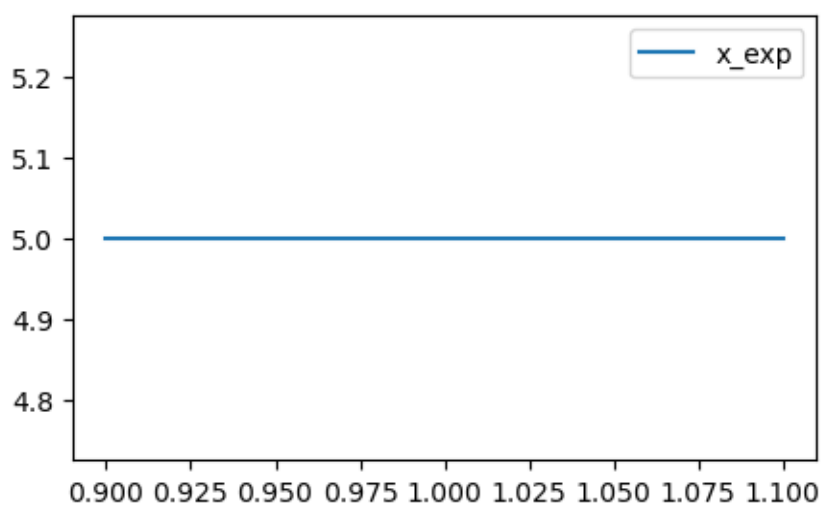
```
# Uncertainty parameters
exp_iv.labels
```

```
['x_exp']
```

```
# Evaluate for a given value of the uncertainty parameters
exp095 = exp_iv.func(x_exp=0.95)
print(
    f"Base value is {exp_base.gdf['value'].sum()}, and the value for x_exp=0.95 is
↪{exp095.gdf['value'].sum()}"
)
```

```
Base value is 657053294559.9105, and the value for x_exp=0.95 is 624200629831.9148
```

```
# Defined distribution
exp_iv.plot(figsize=(5, 3));
```



Example - custom categorical uncertainty parameter

Suppose we want to test different exponents (m=1,2 ; n=1,2) for the LitPop exposure for the country Switzerland.

```
from climada.entity import LitPop

m_min, m_max = (1, 2)
n_min, n_max = (1, 2)


# Define the function
# Note that this here works, but might be slow because the method LitPop is called␣
↪everytime the the function
# is evaluated, and LitPop is relatively slow.
def litpop_cat(m, n):
    exp = Litpop.from_countries("CHE", res_arcsec=150, exponent=[m, n])
    return exp
```

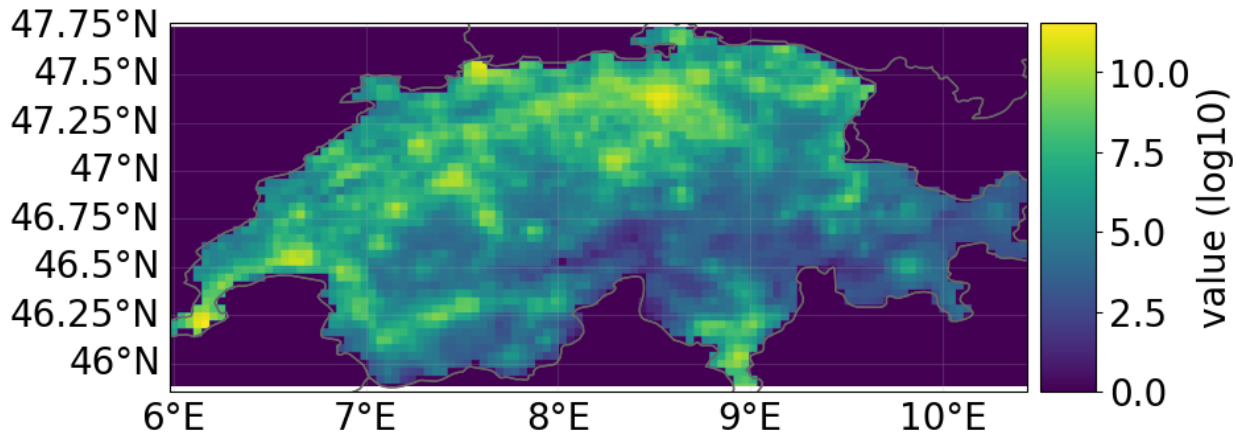
```
# A faster method would be to first create a dictionnary with all the exposures. This␣
↪however
# requires more memory and precomputation time (here ~3-4mins)
exp = LitPop()
litpop_dict = {}
for m in range(m_min, m_max + 1):
    for n in range(n_min, n_max + 1):
        exp_mn = LitPop.from_countries("CHE", res_arcsec=150, exponents=[m, n])
        litpop_dict[(m, n)] = exp_mn


def litpop_cat(m, n, litpop_dict=litpop_dict):
    return litpop_dict[(m, n)]
```

```
# Define the distribution dictionnary
import scipy as sp
from climada.engine.unsequa import InputVar

distr_dict = {
    "m": sp.stats.randint(low=m_min, high=m_max + 1),
    "n": sp.stats.randint(low=n_min, high=n_max + 1),
}

cat_iv = InputVar(
    litpop_cat, distr_dict
)  # One can use either of the above definitions of litpop_cat
```

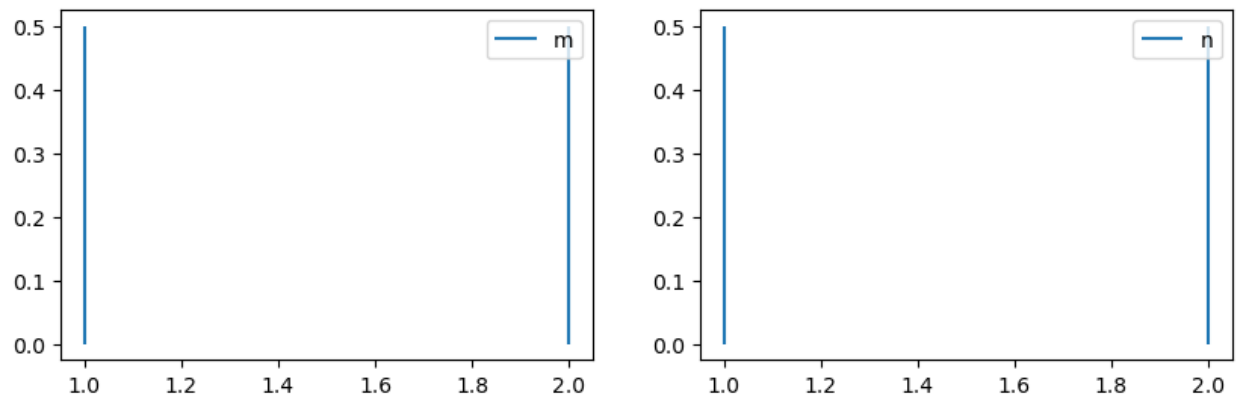
```
# Uncertainty parameters
cat_iv.labels
```

```
['m', 'n']
```

```
cat_iv.evaluate(m=1, n=2).plot_raster();
```




```
cat_iv.plot(figsize=(10, 3));
```



UncOutput

The `UncOutput` class is used to store data from sampling, uncertainty and sensitivity analysis. An UncOutput object can be saved and loaded from .hdf5. The classes `UncImpactOuput` and `UncCostBenefitOutput` are extensions of `UncOutput` specific for `CalcImpact` and `CalcCostBenefit`, respectively.

Data attributes

Attribute	Type	Description
samples_df	pandas.dataframe	Each row represents a sample obtained from the input parameters (one per column) distributions
UncImpactOutput		
aai_agg_unc_df	pandas.dataframe	Uncertainty data for `aai_agg`
freq_curve_unc_df	pandas.dataframe	Uncertainty data for `freq_curve`. One return period per column.
eai_exp_unc_df	pandas.dataframe	Uncertainty data for `eai_exp`. One exposure point per column.
at_event_unc_df	pandas.dataframe	Uncertainty data for `at_event`. One event per column.
UncCostBenefitOutput		
imp_meas_present_un	pandas.dataframe	Uncertainty data for `imp_meas_present`. One measure per column.
imp_meas_future_unc	pandas.dataframe	Uncertainty data for `imp_meas_present`. One measure per column
tot_climate_risk_unc_	pandas.dataframe	Uncertainty data for `tot_climate_risk`. One measure per column.
benefit_unc_df	pandas.dataframe	Uncertainty data for `benefit`. One measure per column.
cost_ben_ratio_unc_df	pandas.dataframe	Uncertainty data for `cost_ben_ratio`. One measure per column.
cost_benefit_kwargs	dictionary	Keyword arguments for `climada.engine.cost_benefit.calc`.

Metadata and input data attributes

These attributes are used for book-keeping and characterize the sample, uncertainty and sensitivity data. These attributes are set by the methods from classes `CalcImpact` and `CalcCostBenefit` used to generate sample, uncertainty and sensitivity data.

Attribute	Type	Description
sampling_met	str	The sampling method as defined in SALib. Possible choices: 'saltelli', 'fast_sampler', 'latin', 'morris', 'dgsm', 'ff'
sampling_kwa	dict	Keyword arguments for the sampling_method.
n_samples	int	Effective number of samples (number of rows of samples_df)
param_lab	list(st	Name of all the uncertainty input parameters
problem_sa	dict	The description of the uncertainty variables and their distribution as used in SALib.
sensitivity_metho	str	Sensitivity analysis method from SALib.analyse Possible choices: 'fast', 'rbd_fact', 'morris', 'sobol', 'delta', 'ff'. Note that in Salib, sampling methods and sensitivity analysis methods should be used in specific pairs.
sensitivity_kwarg	dict	Keyword arguments for sensitivity_method.
unit	str	Unit of the exposures value

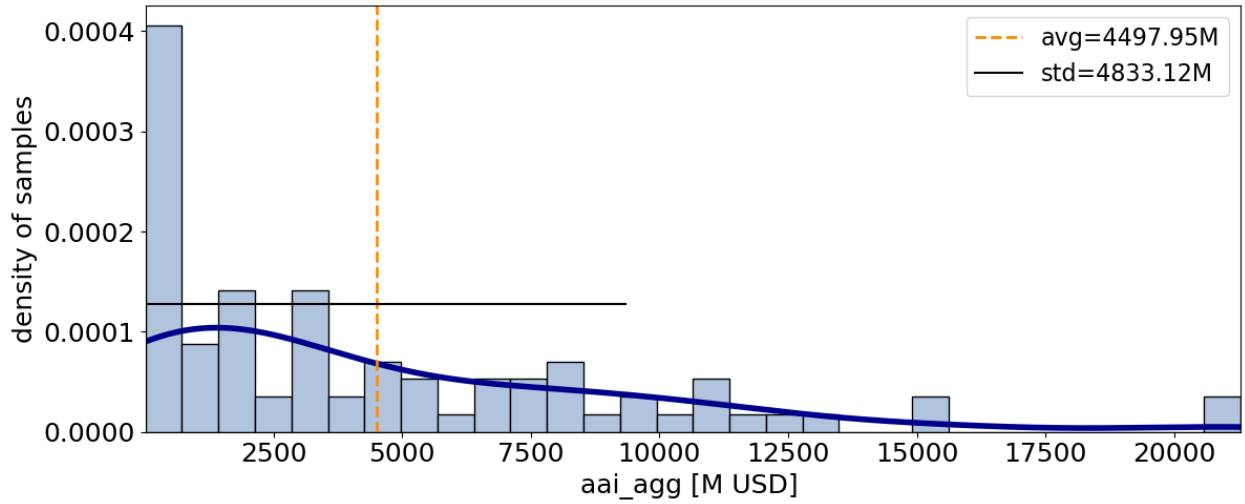
Example from file

Here we show an example loaded from file. In the sections below this class is extensively used and further examples can be found.

```
# Download the test file from the API
# Requires internet connection
from climada.util.constants import TEST_UNC_OUTPUT_IMPACT
from climada.util.api_client import Client

apiclient = Client()
ds = apiclient.get_dataset_info(name=TEST_UNC_OUTPUT_IMPACT, status="test_dataset")
_target_dir, [filename] = apiclient.download_dataset(ds)
```

```
# If you produced your own data, you do not need the API. Just replace 'filename'␣
↪with the path to your file.
from climada.engine.unsequa import UncOutput

unc_imp = UncOutput.from_hdf5(filename)
```

```
unc_imp.plot_uncertainty(metric_list=["aai_agg"], figsize=(12, 5));
```




```
# Download the test file from the API
# Requires internet connection
from climada.util.constants import TEST_UNC_OUTPUT_COSTBEN
from climada.util.api_client import Client

apiclient = Client()
ds = apiclient.get_dataset_info(name=TEST_UNC_OUTPUT_COSTBEN, status="test_dataset")
_target_dir, [filename] = apiclient.download_dataset(ds)
```

```
# If you produced your own data, you do not need the API. Just replace 'filename'␣
↪with the path to your file.
from climada.engine.unsequa import UncOutput

unc_cb = UncOutput.from_hdf5(filename)
```

```
unc_cb.get_uncertainty().tail()
```

```
    Mangroves Benef  Beach nourishment Benef  Seawall Benef  \
35     2.375510e+08             1.932608e+08  234557.682554
36     9.272772e+07             7.643803e+07    9554.257314
37     1.464219e+08             1.179927e+08  192531.748810
38     9.376369e+07             7.722882e+07   10681.112247
39     9.376369e+07             7.722882e+07   10681.112247

    Building code Benef  Mangroves CostBen  Beach nourishment CostBen  \
35         1.584398e+08           6.347120                  10.277239
36         5.501366e+07          16.260133                  25.984286
37         8.979471e+07          10.297402                  16.833137
38         5.555413e+07          12.965484                  20.736269
39         5.555413e+07          16.080478                  25.718218

    Seawall CostBen  Building code CostBen  no measure - risk - future  \
35     4.350910e+04              66.742129                6.337592e+08
36     1.068151e+06             192.217876                2.200547e+08
37     5.300629e+04             117.764285                3.591788e+08
```

(continues on next page)

(continued from previous page)

```
38     7.703765e+05             153.475031                2.222165e+08
39     9.554617e+05             190.347852                2.222165e+08

    no measure - risk_transf - future  ...  \
35                               0.0  ...
36                               0.0  ...
37                               0.0  ...
38                               0.0  ...
39                               0.0  ...

    Beach nourishment - cost_ins - future  Seawall - risk - future  \
35                                      1             6.335246e+08
36                                      1             2.200451e+08
37                                      1             3.589863e+08
38                                      1             2.222058e+08
39                                      1             2.222058e+08

    Seawall - risk_transf - future  Seawall - cost_meas - future  \
35                               0                  1.020539e+10
36                               0                  1.020539e+10
37                               0                  1.020539e+10
38                               0                  8.228478e+09
39                               0                  1.020539e+10

    Seawall - cost_ins - future  Building code - risk - future  \
35                            1                   4.753194e+08
36                            1                   1.650410e+08
37                            1                   2.693841e+08
38                            1                   1.666624e+08
39                            1                   1.666624e+08

    Building code - risk_transf - future  Building code - cost_meas - future  \
35                                     0                        1.057461e+10
36                                     0                        1.057461e+10
37                                     0                        1.057461e+10
38                                     0                        8.526172e+09
39                                     0                        1.057461e+10

    Building code - cost_ins - future  tot_climate_risk
35                                  1      6.337592e+08
36                                  1      2.200547e+08
37                                  1      3.591788e+08
38                                  1      2.222165e+08
39                                  1      2.222165e+08

[5 rows x 29 columns]
```

CalcImpact

Set the InputVars

In this example, we model the impact function for tropical cyclones on the parametric function suggested in Emanuel (2015) with 4 parameters. The exposures total value varies between 80% and 120%. For that hazard, we assume to have no good error estimate and thus do not define an InputVar for the hazard.

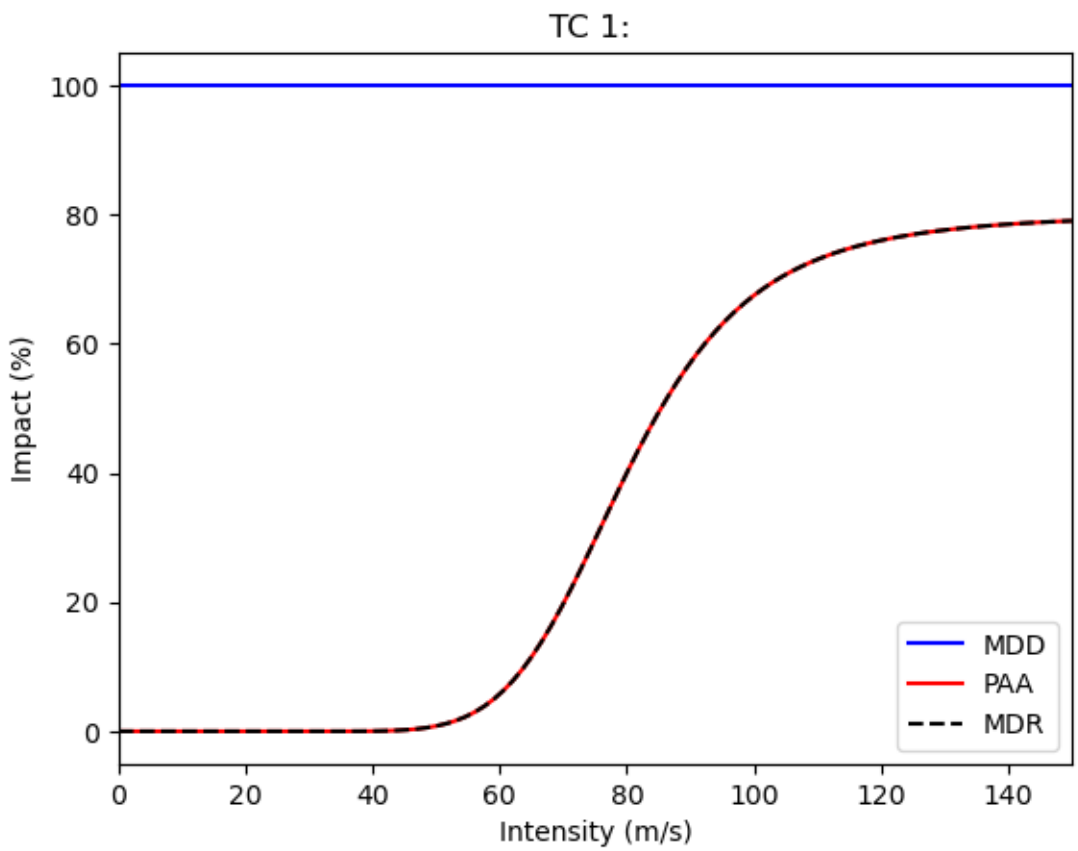
```
# Define the input variable functions
import numpy as np

from climada.entity import ImpactFunc, ImpactFuncSet, Exposures
from climada.util.constants import EXP_DEMO_H5, HAZ_DEMO_H5
from climada.hazard import Hazard


def impf_func(G=1, v_half=84.7, vmin=25.7, k=3, _id=1):

    def xhi(v, v_half, vmin):
        return max([(v - vmin), 0]) / (v_half - vmin)

    def sigmoid_func(v, G, v_half, vmin, k):
        return G * xhi(v, v_half, vmin) ** k / (1 + xhi(v, v_half, vmin) ** k)

    # In-function imports needed only for parallel computing on Windows
    import numpy as np
    from climada.entity import ImpactFunc, ImpactFuncSet

    intensity_unit = "m/s"
    intensity = np.linspace(0, 150, num=100)
    mdd = np.repeat(1, len(intensity))
    paa = np.array([sigmoid_func(v, G, v_half, vmin, k) for v in intensity])
    imp_fun = ImpactFunc("TC", _id, intensity, mdd, paa, intensity_unit)
    imp_fun.check()
    impf_set = ImpactFuncSet([imp_fun])
    return impf_set


haz = Hazard.from_hdf5(HAZ_DEMO_H5)
exp_base = Exposures.from_hdf5(EXP_DEMO_H5)
# It is a good idea to assign the centroids to the base exposures in order to avoid␣
↪repeating this
# potentially costly operation for each sample.
exp_base.assign_centroids(haz)


def exp_base_func(x_exp, exp_base):
    exp = exp_base.copy()
    exp.gdf["value"] *= x_exp
    return exp
```

(continues on next page)

(continued from previous page)

```
from functools import partial

exp_func = partial(exp_base_func, exp_base=exp_base)
```

```
# Visualization of the parametrized impact function
impf_func(G=0.8, v_half=80, vmin=30, k=5).plot();
```



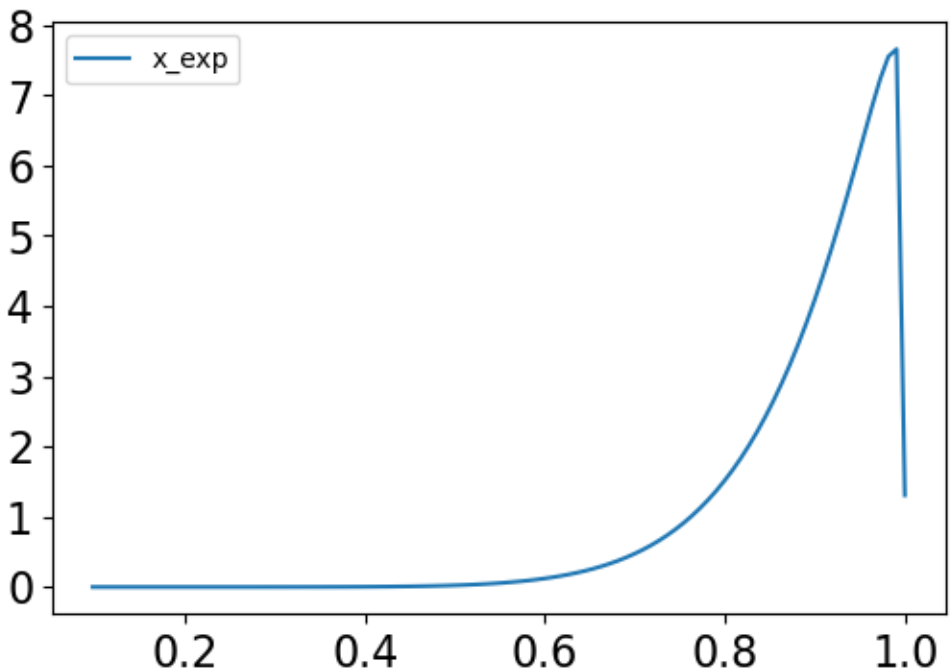

```
# Define the InputVars

import scipy as sp
from climada.engine.unsequa import InputVar

exp_distr = {
    "x_exp": sp.stats.beta(10, 1.1)
}  # This is not really a reasonable distribution but is used
# here to show that you can use any scipy distribution.

exp_iv = InputVar(exp_func, exp_distr)

impf_distr = {
    "G": sp.stats.truncnorm(0.5, 1.5),
    "v_half": sp.stats.uniform(35, 65),
    "vmin": sp.stats.uniform(0, 15),
    "k": sp.stats.uniform(1, 4),
```

(continues on next page)

(continued from previous page)

```
}
impf_iv = InputVar(impf_func, impf_distr)
```

```
import matplotlib.pyplot as plt

ax = exp_iv.plot(figsize=(6, 4))
plt.yticks(fontsize=16)
plt.xticks(fontsize=16);
```



Compute uncertainty and sensitivity using default methods

First, we define the UncImpact object with our uncertainty variables.

```
from climada.engine.unsequa import CalcImpact

calc_imp = CalcImpact(exp_iv, impf_iv, haz)
```

Next, we generate samples for the uncertainty parameters using the default methods. Note that depending on the chosen Salib method, the effective number of samples differs from the input variable N. For the default 'saltelli', with `calc_second_order=True`, the effective number is N(2D+2), with D the number of uncertainty parameters. See SAlib for more information.

```
output_imp = calc_imp.make_sample(N=2**7, sampling_kwargs={"skip_values": 2**8})
output_imp.get_samples_df().tail()
```

```
         x_exp         G     v_half       vmin         k
1531  0.876684  1.242977  53.662109   2.080078  4.539062
```

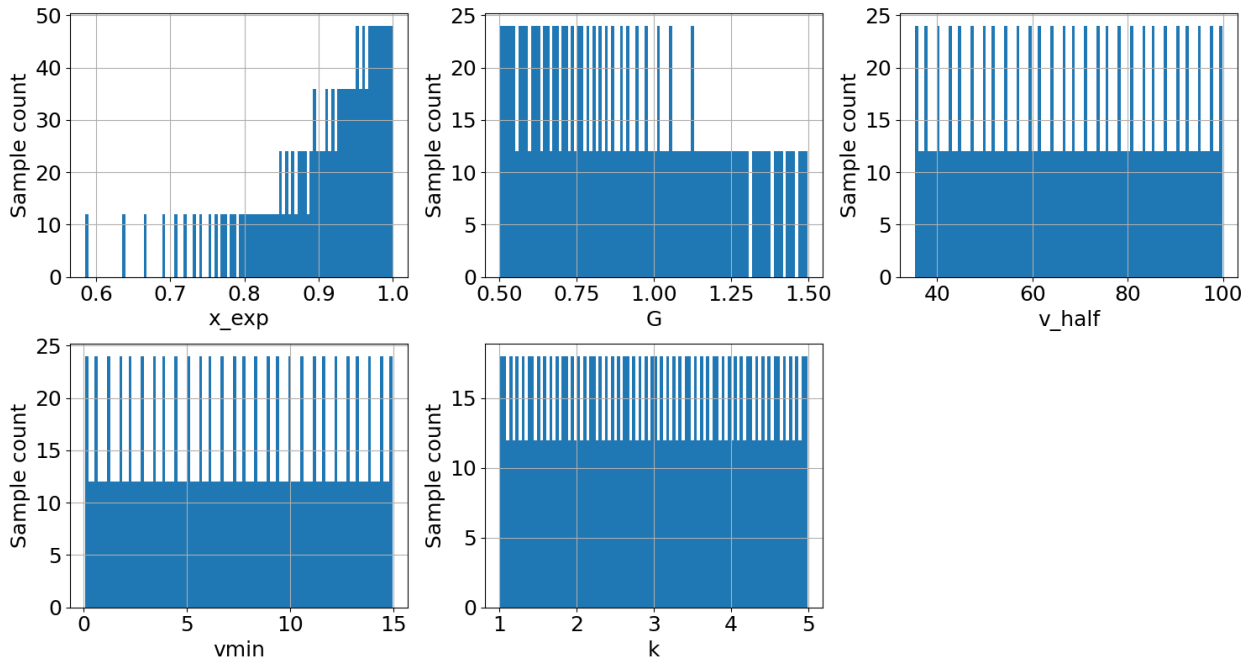
(continues on next page)

(continued from previous page)

```
1532  0.876684  0.790617  44.013672   2.080078  4.539062
1533  0.876684  0.790617  53.662109  13.681641  4.539062
1534  0.876684  0.790617  53.662109   2.080078  3.960938
1535  0.876684  0.790617  53.662109   2.080078  4.539062
```

The resulting samples can be visualized in plots.

```
output_imp.plot_sample(figsize=(15, 8));
```



Now we can compute the value of the impact metrics for all the samples. In this example, we additionaly chose to restrict the return periods 50, 100, and 250 years. By default, `eai_exp` and `at_event` are not stored.

```
output_imp = calc_imp.uncertainty(output_imp, rp=[50, 100, 250])
```

The distributions of metrics ouputs are stored as dictionaries of pandas dataframe. The metrics are directly taken from the output of `climada.impact.calc`. For each metric, on dataframe is made.

```
# All the computed uncertainty metrics attribute
output_imp.uncertainty_metrics
```

```
['aai_agg', 'freq_curve']
```

```
# One uncertainty dataframe
output_imp.get_unc_df("aai_agg").tail()
```

```
           aai_agg
1531  2.905571e+09
1532  3.755172e+09
1533  1.063119e+09
1534  2.248718e+09
1535  1.848139e+09
```

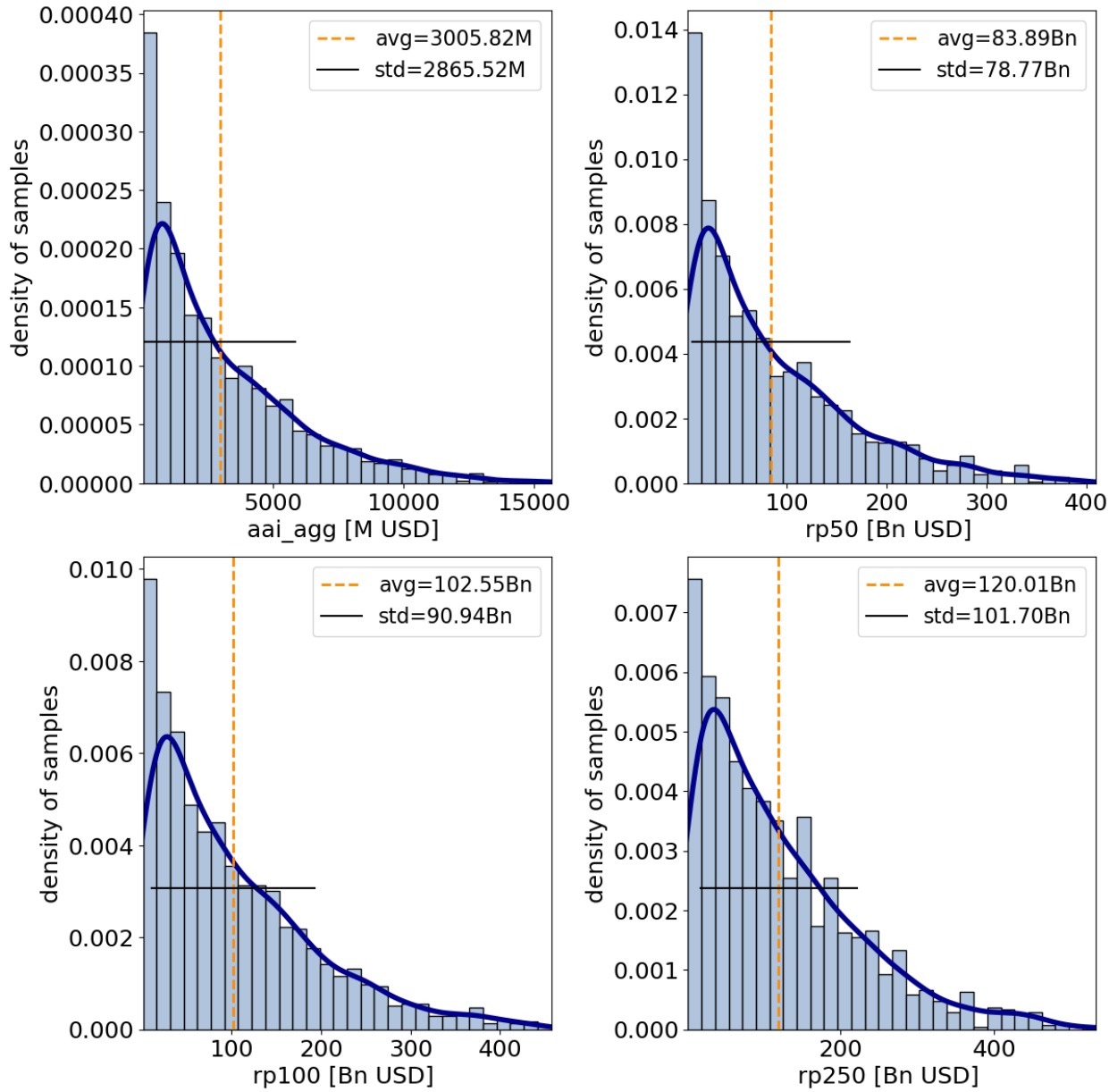
Accessing the uncertainty is in general done via the method `get_uncertainty()`. If none are specified, all metrics are returned.

```
output_imp.get_uncertainty().tail()
```

```
           aai_agg          rp50         rp100         rp250
1531  2.905571e+09  8.324391e+10  1.162643e+11  1.510689e+11
1532  3.755172e+09  1.096005e+11  1.460838e+11  1.809413e+11
1533  1.063119e+09  2.892734e+10  4.720869e+10  6.807561e+10
1534  2.248718e+09  6.468855e+10  8.653474e+10  1.085266e+11
1535  1.848139e+09  5.294874e+10  7.395191e+10  9.609003e+10
```

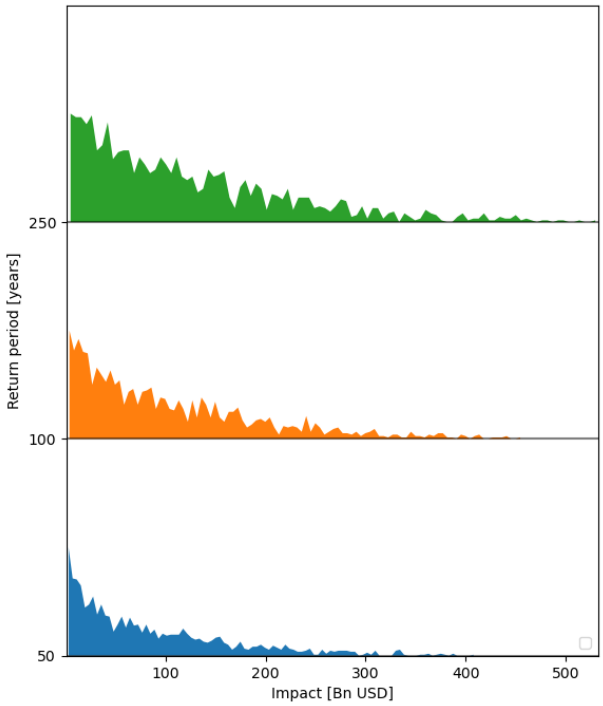
The distributions of the one-dimensioanl metrics (`eai_exp` and `at_event` are never shown with this method) can be vizualised with plots.

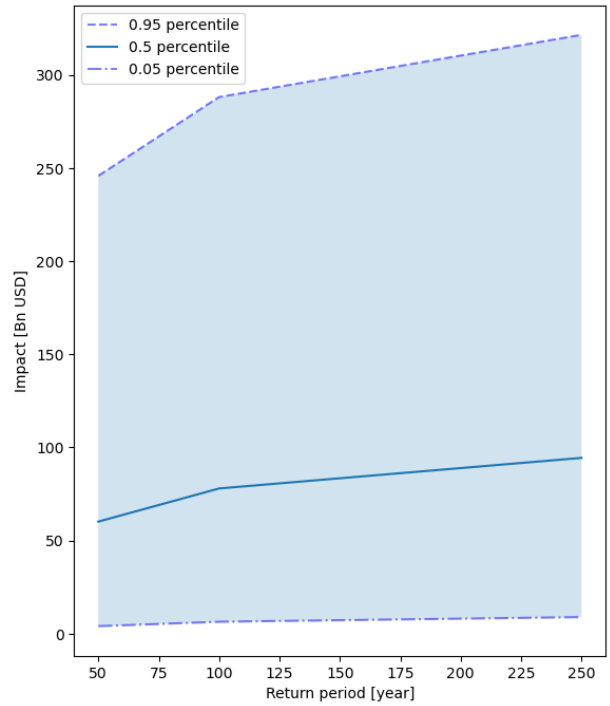
```
output_imp.plot_uncertainty(figsize=(12, 12));
```




```
# Specific plot for the return period distributions
output_imp.plot_rp_uncertainty(figsize=(14.3, 8));
```

```
No artists with labels found to put in legend.  Note that artists whose label start
↪with an underscore are ignored when legend() is called with no argument.
```





Now that a distribution of the impact metrics has been computed for each sample, we can also compute the sensitivity indices for each metrics to each uncertainty parameter. Note that the chosen method for the sensitivity analysis should correpond to its sampling partner as defined in the SAlib package.

The sensitivity indices dictionnaries outputs from the SAlib methods are stored in the same structure of nested dictionnaries as the metrics distributions. Note that depending on the chosen sensitivity analysis method the returned indices dictionnary will return specific types of sensitivity indices with specific names. Please get familiar with SAlib for more information.

Note that in our case, several of the second order sensitivity indices are negative. For the default method `sobol`, this indicates that the algorithm has not converged and cannot give realiable values for these sensitivity indices. If this happens, please use a larger number of samples. Here we will focus on the first-order indices.

```
output_imp = calc_imp.sensitivity(output_imp)
```

Similarly to the uncertainty case, the data is stored in dataframe attributes.

```
output_imp.sensitivity_metrics
```

```
['aai_agg', 'freq_curve']
```

```
output_imp.get_sens_df("aai_agg").tail()
```

```
         si param  param2  aai_agg
65  S2_conf     k   x_exp      NaN
66  S2_conf     k       G      NaN
67  S2_conf     k  v_half      NaN
68  S2_conf     k    vmin      NaN
69  S2_conf     k       k      NaN
```

To obtain the sensitivity interms of a particular sensitivity index, use the method `get_sensisitivity()`. If none is specified, the value of the index for all metrics is returned.

```
output_imp.get_sensitivity("S1")
```

```
   si   param param2   aai_agg      rp50     rp100     rp250
0  S1   x_exp   None  0.001040  0.000993  0.000930  0.001150
1  S1       G   None  0.073408  0.075781  0.084662  0.093718
2  S1  v_half   None  0.514220  0.553640  0.596659  0.619366
3  S1    vmin   None  0.012642  0.014407  0.012068  0.010065
4  S1       k   None  0.213491  0.189862  0.134867  0.095861
```

Sometimes, it is useful to simply know what is the largest sensitivity index for each metric.

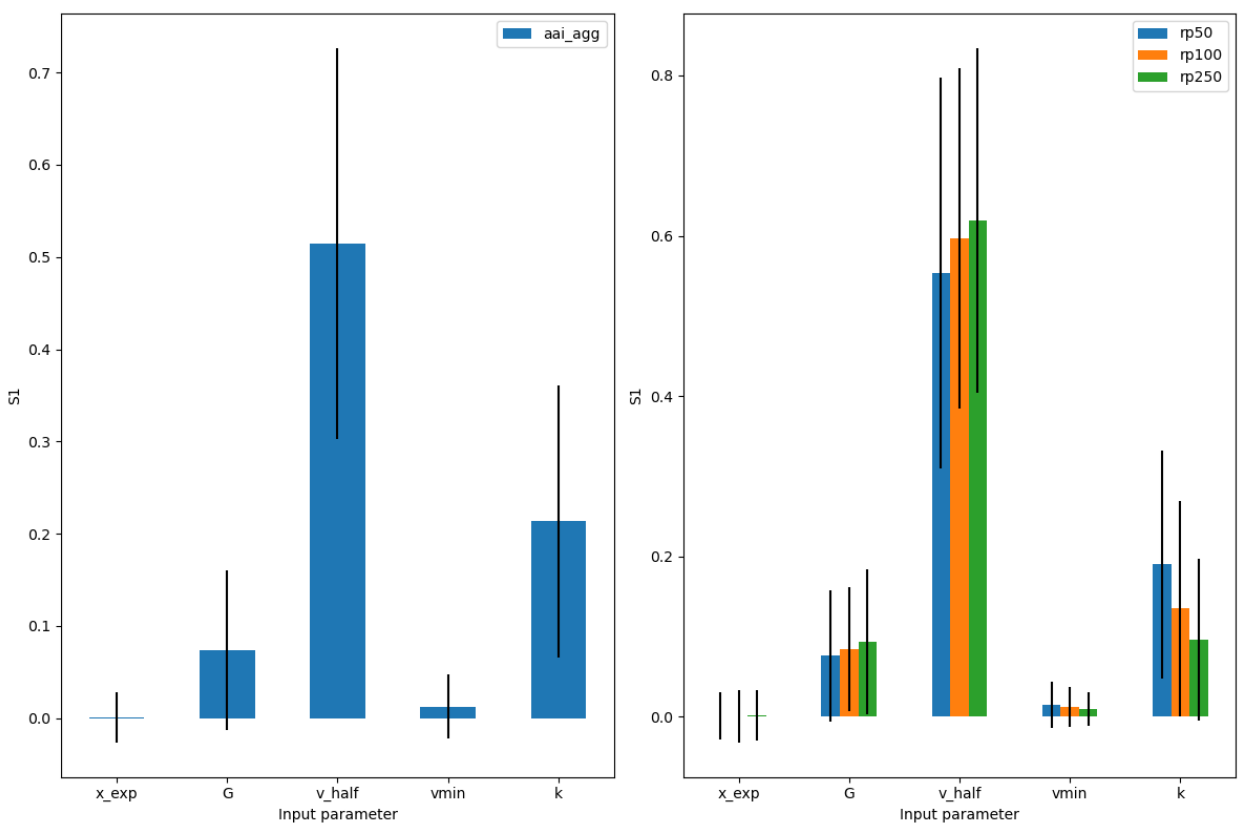
```
output_imp.get_largest_si(salib_si="S1")
```

```
    metric   param param2        si
0  aai_agg  v_half   None  0.514220
1     rp50  v_half   None  0.553640
2    rp100  v_half   None  0.596659
3    rp250  v_half   None  0.619366
```

The value of the sensitivity indices can be plotted for each metric that is one-dimensional (`eai_exp` and `at_event` are not shown in this plot).

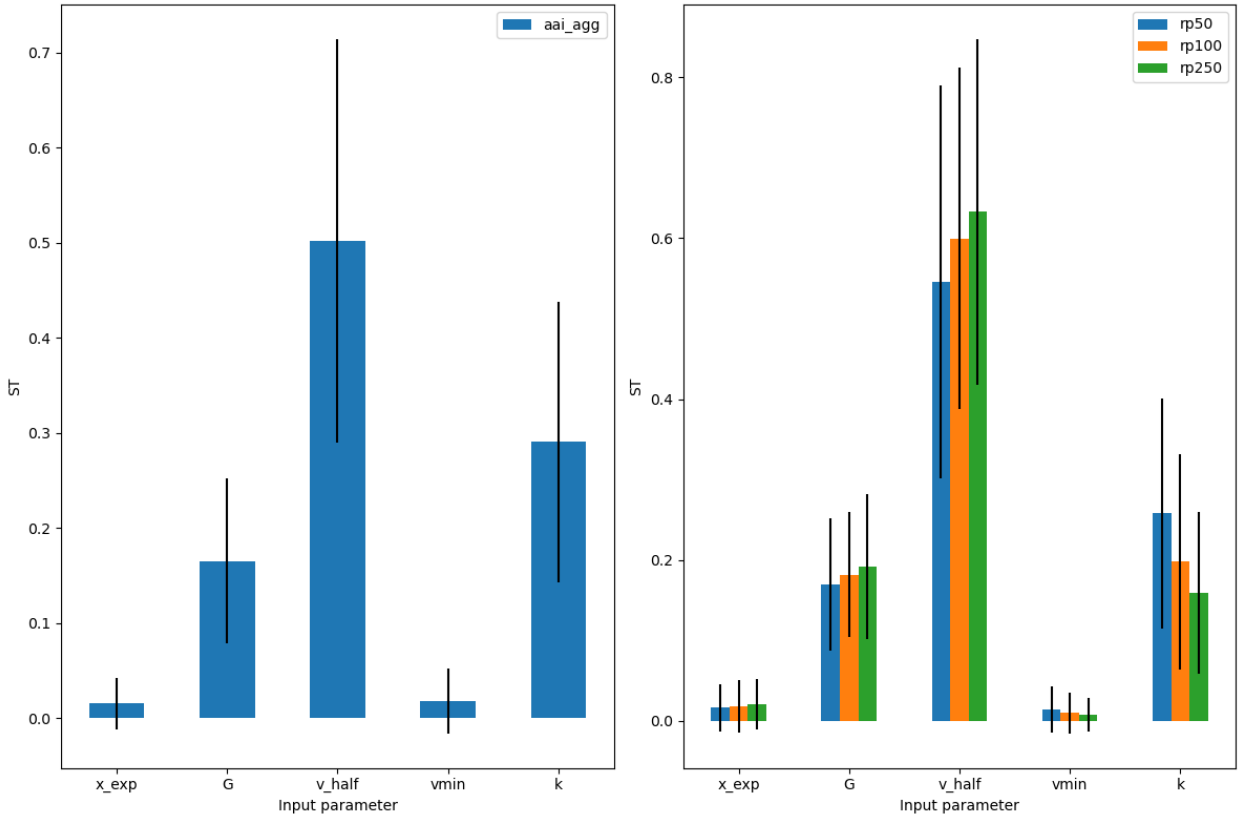
We see that both the errors in `freq_curve` and in `aai_agg` are mostly determined by `x_exp` and `v_half`. Finally, we see small differences in the sensitivity of the different return periods.

```
# Default for 'sobol' is to plot 'S1' sensitivity index.
output_imp.plot_sensitivity(figsize=(12, 8));
```



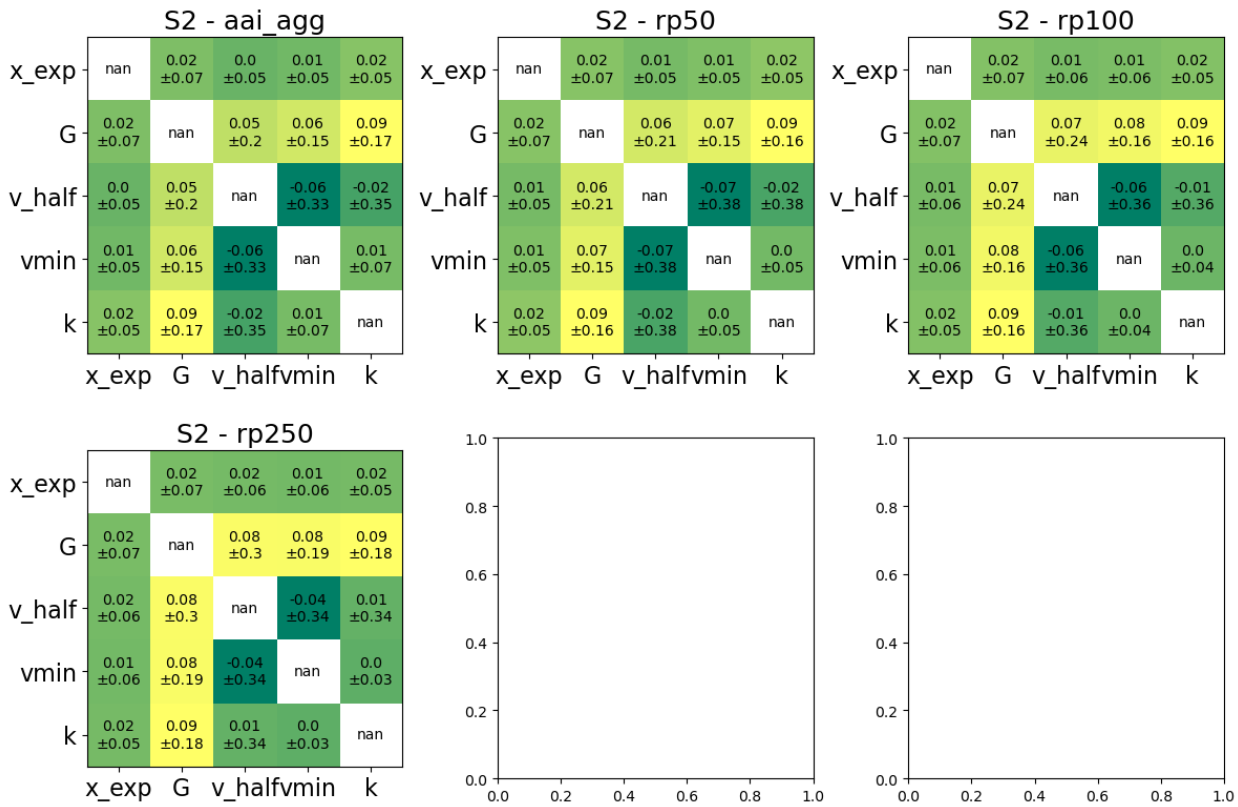
Note that since we have quite a few measures, the imp_meas_fut and imp_meas_pres plots are too crowded. We can select only the other metrics easily. In addition, instead of showing first order sensitivity 'S1', we can plot the total sensitivity 'ST'.

```
output_imp.plot_sensitivity(salib_si="ST", figsize=(12, 8));
```



One can also vizualise the second-order sensitivity indices in the form of a correlation matrix.

```
output_imp.plot_sensitivity_second_order(figsize=(12, 8));
```

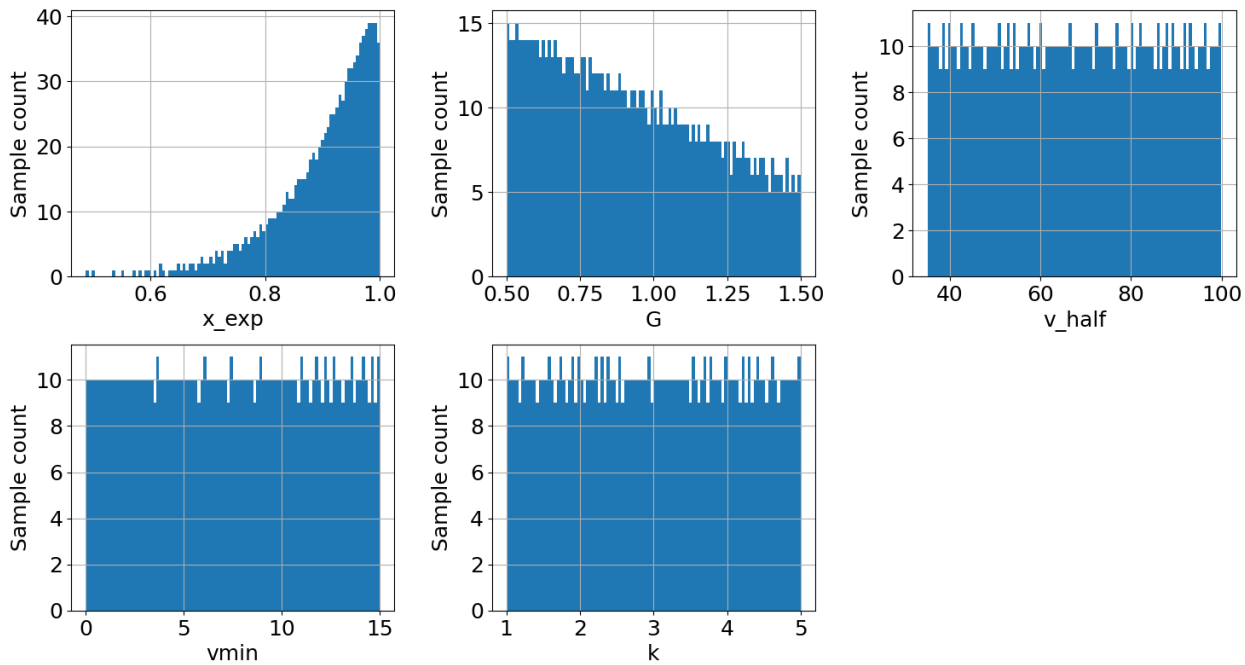


A few non-default parameters

We shall use the same uncertainty variables as in the previous section but show a few possibilities to use non-default method arguments.

```
# Sampling method "latin" hypercube instead of `saltelli`.
from climada.engine.unsequa import CalcImpact

calc_imp2 = CalcImpact(exp_iv, impf_iv, haz)
output_imp2 = calc_imp2.make_sample(N=1000, sampling_method="latin")
```

```
output_imp2.plot_sample(figsize=(15, 8));
```




```
# Compute also the distribution of the metric `eai_exp`
# To speed-up the comutations, we can use more than one process
# Note that for large dataset a single process might be more efficient
import time

calc_imp2 = CalcImpact(exp_iv, impf_iv, haz)
output_imp2 = calc_imp2.make_sample(N=1000, sampling_method="latin")

start = time.time()
output_imp2 = calc_imp2.uncertainty(
    output_imp2, rp=[50, 100, 250], calc_eai_exp=True, calc_at_event=True, processes=4
)
end = time.time()
time_passed = end - start
print(f"Time passed with pool: {time_passed}")
```

```
Time passed with pool: 2.8349649906158447
```

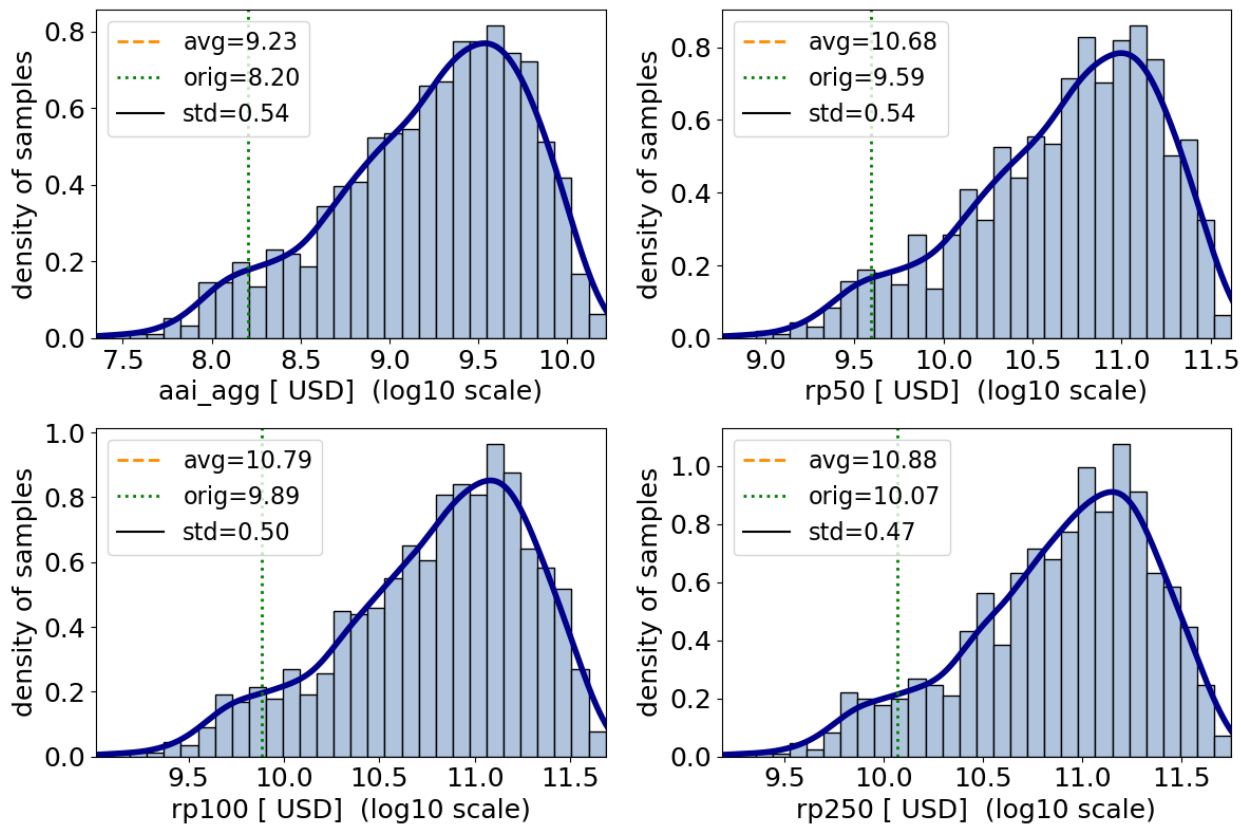
```
from climada.engine.unsequa import CalcImpact
import time
```

```
calc_imp2 = CalcImpact(exp_iv, impf_iv, haz)
output_imp2 = calc_imp2.make_sample(N=1000, sampling_method="latin")

start2 = time.time()
output_imp2 = calc_imp2.uncertainty(
    output_imp2, rp=[50, 100, 250], calc_eai_exp=True, calc_at_event=True
)
end2 = time.time()
time_passed_nopool = end2 - start2
print(f"Time passed without pool: {time_passed_nopool}")
```

```
Time passed without pool: 8.287853956222534
```

```
# Add the original value of the impacts (without uncertainty) to the uncertainty plot
from climada.engine import ImpactCalc

imp = ImpactCalc(exp_base, impf_func(), haz).impact(assign_centroids=False)
aai_agg_o = imp.aai_agg
freq_curve_o = imp.calc_freq_curve([50, 100, 250]).impact
orig_list = [aai_agg_o] + list(freq_curve_o) + [1]
```

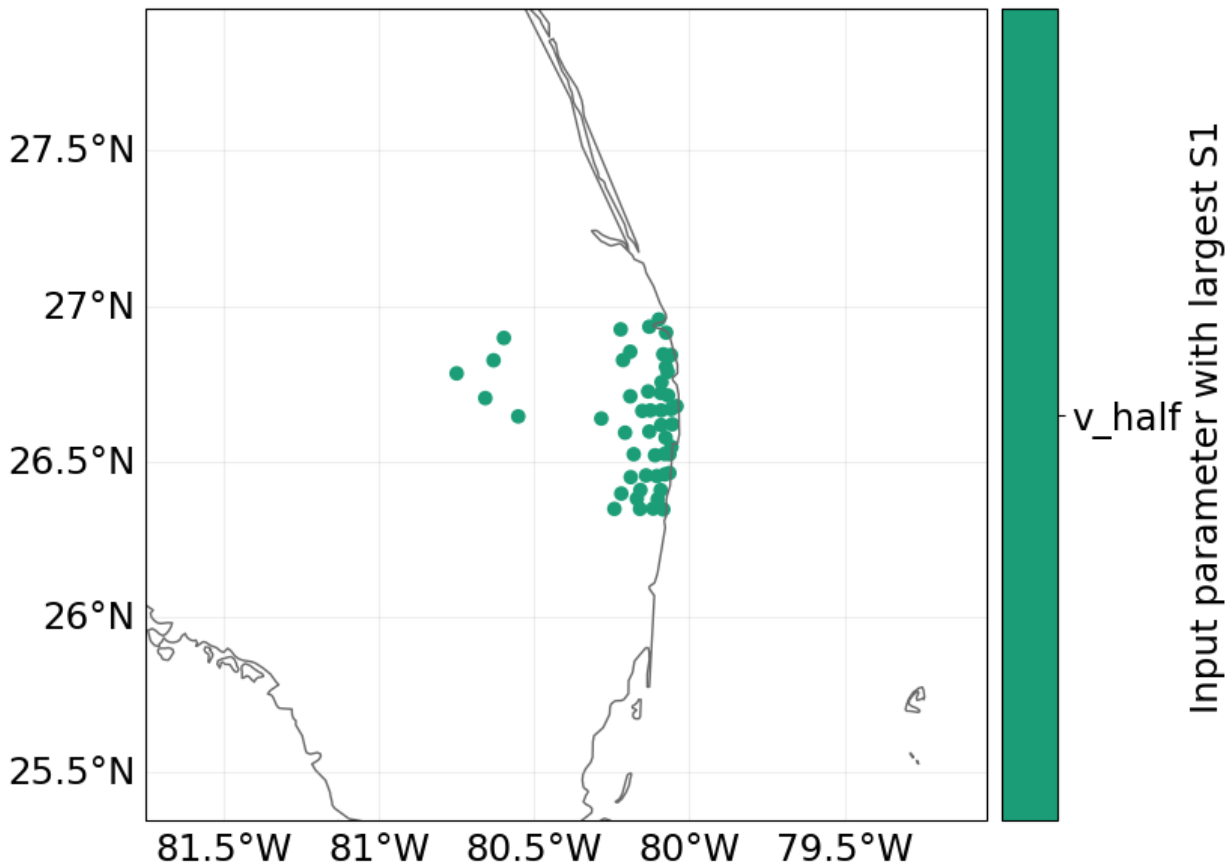
```
# plot the aai_agg and freq_curve uncertainty only
# use logarithmic x-scale
output_imp2.plot_uncertainty(
    metric_list=["aai_agg", "freq_curve"],
    orig_list=orig_list,
    log=True,
    figsize=(12, 8),
);
```




```
# Use the method 'rbd_fast' which is recommend in pair with 'latin'. In addition,␣
↪change one of the kwargs
# (M=15) of the salib sampling method.
output_imp2 = calc_imp2.sensitivity(
    output_imp2, sensitivity_method="rbd_fast", sensitivity_kwargs={"M": 15}
)
```

Since we computed the distribution and sensitivity indices for the total impact at each exposure point, we can plot a map of the largest sensitivity index in each exposure location. For every location, the most sensitive parameter is `v_half`, meaning that the average annual impact at each location is most sensitivity to the ucnertainty in the impact function slope scaling parameter.

```
output_imp2.plot_sensitivity_map();
```




```
output_imp2.get_largest_si(salib_si="S1", metric_list=["eai_exp"]).tail()
```

```
   metric   param param2        si
45     45  v_half   None  0.471587
46     46  v_half   None  0.471587
47     47  v_half   None  0.471587
48     48  v_half   None  0.467530
49     49  v_half   None  0.471587
```

CalcDeltaImpact

The main goal of this class is to perform an uncertainty and sensitivity analysis of the "delta" impact between a reference state and future (or any other "to be compared") state.

Classical example: risk increase in the future with climate change and socio economic development. In this case, the uncertainty and sensitivity analysis in performed on the estimated risk (delta) increase in the future relative to the present-day baseline.

The uncertainty and sensitivity analysis for CalcDeltaImpact is completely analogous to the Impact case. It is slightly more complex as there are more input variables.

Note, the logic of this class works with any comparison between an initial (reference and final (altered) risk or impact state and is not limited to the scope of climate change and socio-economic development in the future.

Set the Input Vars

We'll work through an analogous example as in CalcImpact next.

```
import numpy as np

from climada.entity import ImpactFunc, ImpactFuncSet, Exposures
from climada.util.constants import EXP_DEMO_H5, HAZ_DEMO_H5
from climada.hazard import Centroids, TCTracks, Hazard, TropCyclone


def impf_func(G=1, v_half=84.7, vmin=25.7, k=3, _id=1):

    def xhi(v, v_half, vmin):
        return max([(v - vmin), 0]) / (v_half - vmin)

    def sigmoid_func(v, G, v_half, vmin, k):
        return G * xhi(v, v_half, vmin) ** k / (1 + xhi(v, v_half, vmin) ** k)

    # In-function imports needed only for parallel computing on Windows
    intensity_unit = "m/s"
    intensity = np.linspace(0, 150, num=100)
    mdd = np.repeat(1, len(intensity))
    paa = np.array([sigmoid_func(v, G, v_half, vmin, k) for v in intensity])
    imp_fun = ImpactFunc("TC", _id, intensity, mdd, paa, intensity_unit)
    imp_fun.check()
    impf_set = ImpactFuncSet([imp_fun])
    return impf_set
```

Load the hazard set and apply climate change factors to it. This yields a hazard representation in 2050 under 4 RCP scenarios. For a full documentation of this function please refer to the *TropCyclone* tutorial.

```
# load historical hazard set
haz = TropCyclone.from_hdf5(HAZ_DEMO_H5)
haz.basin = ["NA"] * haz.size

# apply climate change factors
haz_26 = haz.apply_climate_scenario_knu(ref_year=2050, rcp_scenario=26)
haz_45 = haz.apply_climate_scenario_knu(ref_year=2050, rcp_scenario=45)
haz_60 = haz.apply_climate_scenario_knu(ref_year=2050, rcp_scenario=60)
haz_85 = haz.apply_climate_scenario_knu(ref_year=2050, rcp_scenario=85)

# pack future hazard sets into dictionary - we want to sample from this dictionary␣
↪later
haz_fut_list = [haz_26, haz_45, haz_60, haz_85]
tc_haz_fut_dict = {}
```

(continues on next page)

(continued from previous page)

```
for r, rcp in enumerate(["26", "45", "60", "85"]):
    tc_haz_fut_dict[rcp] = haz_fut_list[r]
```

```
exp_base = Exposures.from_hdf5(EXP_DEMO_H5)
# It is a good idea to assign the centroids to the base exposures in order to avoid␣
↪repeating this
# potentially costly operation for each sample.
exp_base.assign_centroids(haz)


def exp_base_func(x_exp, exp_base):
    exp = exp_base.copy()
    exp.gdf["value"] *= x_exp
    return exp


from functools import partial

exp_func = partial(exp_base_func, exp_base=exp_base)
```

```
import scipy as sp
from climada.engine.unsequa import InputVar

exp_distr = {
    "x_exp": sp.stats.beta(10, 1.1)
}  # This is not really a reasonable distribution but is used
# here to show that you can use any scipy distribution.

exp_iv = InputVar(exp_func, exp_distr)

impf_distr = {
    "G": sp.stats.truncnorm(0.5, 1.5),
    "v_half": sp.stats.uniform(35, 65),
    "vmin": sp.stats.uniform(0, 15),
    "k": sp.stats.uniform(1, 4),
}
impf_iv = InputVar(impf_func, impf_distr)
```

Next we define the function for the future hazard representation. It's a simple function that allows us to draw from the hazard dictionary of hazard sets under different RCP scenarios. Note, we do not investigate other hazard related uncertainties in this example.

```
rcp_key = {0: "26", 1: "45", 2: "60", 3: "85"}


# future
def haz_fut_func(rcp_scenario):
    haz_fut = tc_haz_fut_dict[rcp_key[rcp_scenario]]
    return haz_fut


haz_fut_distr = {"rcp_scenario": sp.stats.randint(0, 4)}
```

(continues on next page)

(continued from previous page)

```
haz_fut_iv = InputVar(haz_fut_func, haz_fut_distr)
```

Compute uncertainty and sensitivity

In contrast to CalcImpact, we define InputVars for initial and final states of exposure, impact function, hazard. This class requires 6 input variables. For the sake of simplicity, we did not define varying input variables for the initial and future exposure and vulnerability in the example. Hence, the exp_iv and impf_iv are passed to CalcDeltaImpact twice.

```
from climada.engine.unsequa import CalcDeltaImpact

calc_imp = CalcDeltaImpact(exp_iv, impf_iv, haz, exp_iv, impf_iv, haz_fut_iv)
```

```
2024-01-25 15:36:53,385 - climada.engine.unsequa.calc_base - WARNING -

The input parameter x_exp is shared among at least 2 input variables. Their␣
↪uncertainty is thus computed with the same samples for this input paramter.


2024-01-25 15:36:53,389 - climada.engine.unsequa.calc_base - WARNING -

The input parameter G is shared among at least 2 input variables. Their uncertainty␣
↪is thus computed with the same samples for this input paramter.


2024-01-25 15:36:53,390 - climada.engine.unsequa.calc_base - WARNING -

The input parameter v_half is shared among at least 2 input variables. Their␣
↪uncertainty is thus computed with the same samples for this input paramter.


2024-01-25 15:36:53,393 - climada.engine.unsequa.calc_base - WARNING -

The input parameter vmin is shared among at least 2 input variables. Their␣
↪uncertainty is thus computed with the same samples for this input paramter.


2024-01-25 15:36:53,394 - climada.engine.unsequa.calc_base - WARNING -

The input parameter k is shared among at least 2 input variables. Their uncertainty␣
↪is thus computed with the same samples for this input paramter.
```

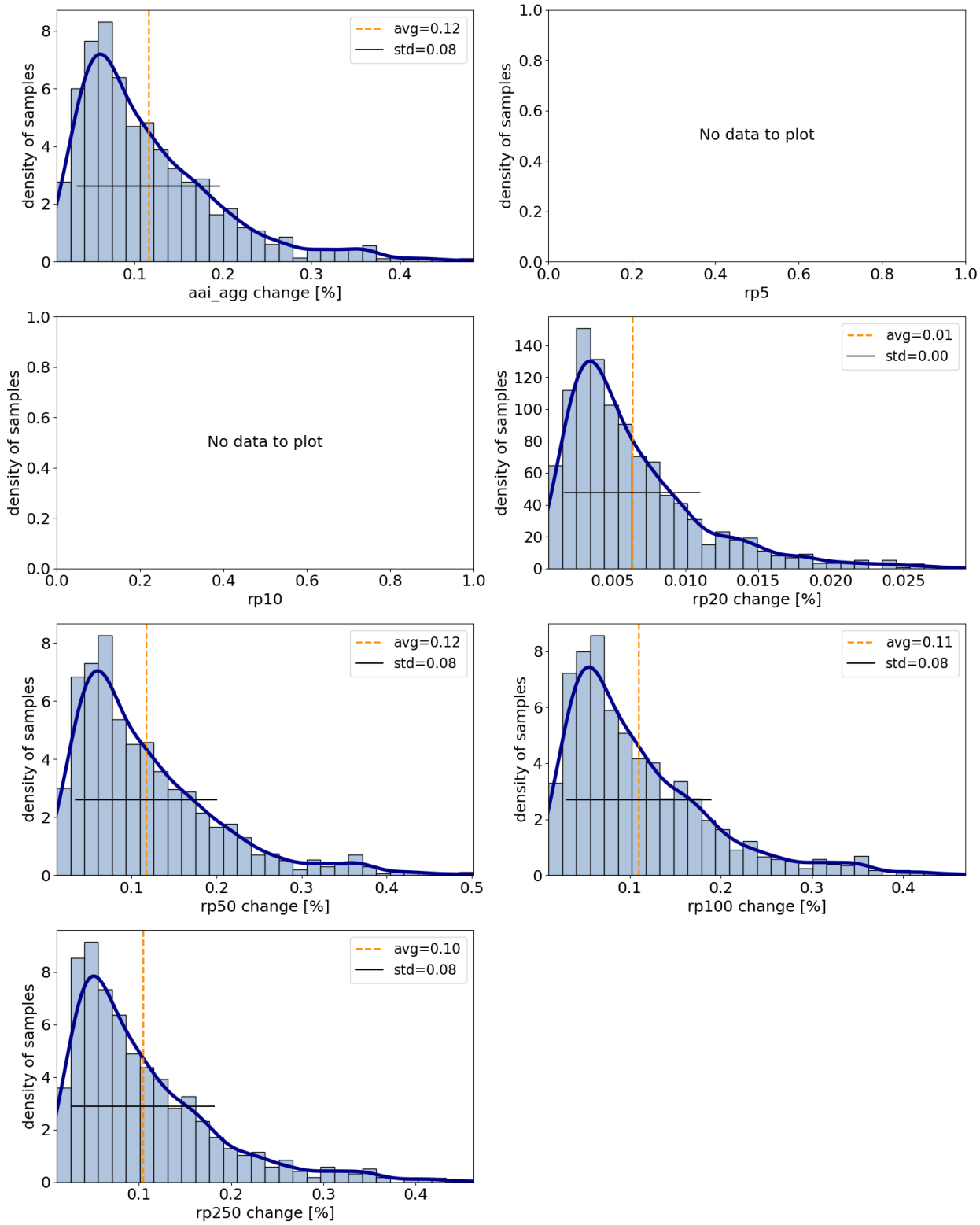
```
output_imp = calc_imp.make_sample(N=2**7)
output_imp.get_samples_df().tail()

output_imp = calc_imp.uncertainty(output_imp)
```

Plotting functionalities work analogous to CalcImpact. By setting `calc_delta=True`, the axis labels are adjusted.

```
from climada.engine.unsequa import UncOutput

output_imp.plot_uncertainty(calc_delta=True)
```

```
No data to plot for 'rp5'.
No data to plot for 'rp10'.
```

```
array([[<Axes: xlabel='aai_agg change [%]', ylabel='density of samples'>,
        <Axes: xlabel='rp5', ylabel='density of samples'>],
       [<Axes: xlabel='rp10', ylabel='density of samples'>,
        <Axes: xlabel='rp20 change [%]', ylabel='density of samples'>],
       [<Axes: xlabel='rp50 change [%]', ylabel='density of samples'>,
        <Axes: xlabel='rp100 change [%]', ylabel='density of samples'>],
       [<Axes: xlabel='rp250 change [%]', ylabel='density of samples'>,
        <Axes: >]], dtype=object)
```

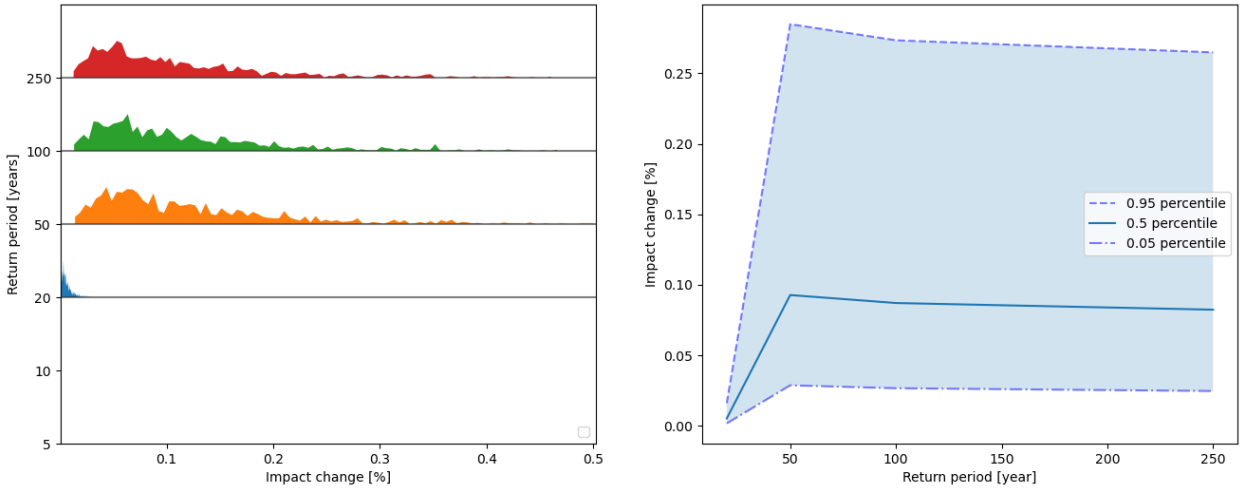



```
from climada.engine.unsequa import UncOutput

output_imp.plot_rp_uncertainty(calc_delta=True)
```

```
No artists with labels found to put in legend.  Note that artists whose label start␣
↪with an underscore are ignored when legend() is called with no argument.
```

```
Skipping plot for 'rp5': insufficient data.
Skipping plot for 'rp10': insufficient data.
```

```
array([<Axes: xlabel='Impact change [%]', ylabel='Return period [years]'>,
       <Axes: xlabel='Return period [year]', ylabel='Impact change [%]'>],
      dtype=object)
```

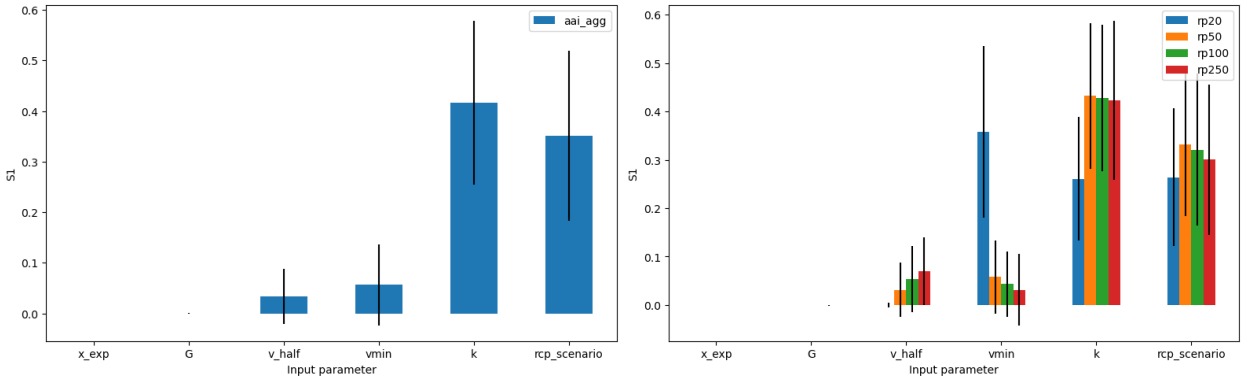



```
# compute sensitivity
output_imp = calc_imp.sensitivity(output_imp)
```

```
# plot sensitivity
output_imp.plot_sensitivity()
```

```
2024-01-25 15:37:27,753 - climada.engine.unsequa.unc_output - WARNING - All-NaN␣
↪columns encountered: ['rp5', 'rp10']
```

```
array([<Axes: xlabel='Input parameter', ylabel='S1'>,
       <Axes: xlabel='Input parameter', ylabel='S1'>], dtype=object)
```



The rest of the functionalities that apply to CalcImpact also work for the CalcDeltaImpact class. Hence, refer to the

sections above for details.

CalcCostBenefit

The uncertainty and sensitivity analysis for CostBenefit is completely analogous to the Impact case. It is slightly more complex as there are more input variables.

Set the Input Vars

```
import copy
from climada.util.constants import ENT_DEMO_TODAY, ENT_DEMO_FUTURE, HAZ_DEMO_H5
from climada.entity import Entity
from climada.hazard import Hazard


# Entity today has an uncertainty in the total asset value
def ent_today_func(x_ent):
    # In-function imports needed only for parallel computing on Windows
    from climada.entity import Entity
    from climada.util.constants import ENT_DEMO_TODAY

    entity = Entity.from_excel(ENT_DEMO_TODAY)
    entity.exposures.ref_year = 2018
    entity.exposures.gdf["value"] *= x_ent
    return entity


# Entity in the future has a +- 10% uncertainty in the cost of all the adapatation␣
↪measures
def ent_fut_func(m_fut_cost):
    # In-function imports needed only for parallel computing on Windows
    from climada.entity import Entity
    from climada.util.constants import ENT_DEMO_FUTURE

    entity = Entity.from_excel(ENT_DEMO_FUTURE)
    entity.exposures.ref_year = 2040
    for meas in entity.measures.get_measure("TC"):
        meas.cost *= m_fut_cost
    return entity


haz_base = Hazard.from_hdf5(HAZ_DEMO_H5)


# The hazard intensity in the future is also uncertainty by a multiplicative factor
def haz_fut(x_haz_fut, haz_base):
    # In-function imports needed only for parallel computing on Windows
    import copy
    from climada.hazard import Hazard
    from climada.util.constants import HAZ_DEMO_H5
```

(continues on next page)

(continued from previous page)

```
    haz = copy.deepcopy(haz_base)
    haz.intensity = haz.intensity.multiply(x_haz_fut)
    return haz


from functools import partial

haz_fut_func = partial(haz_fut, haz_base=haz_base)
```

Check that costs for measures are changed as desired.

```
costs_1 = [meas.cost for meas in ent_fut_func(1).measures.get_measure("TC")]
costs_05 = [meas.cost for meas in ent_fut_func(0.5).measures.get_measure("TC")]
print(
    f"\nThe cost for m_fut_cost=1 are {costs_1}\n"
    f"The cost for m_fut_cost=0.5 are {costs_05}"
);
```

```
The cost for m_fut_cost=1 are [1311768360.8515418, 1728000000.0, 8878779433.630093,␣
↪9200000000.0]
The cost for m_fut_cost=0.5 are [655884180.4257709, 864000000.0, 4439389716.815046,␣
↪4600000000.0]
```

Define the InputVars

```
import scipy as sp
from climada.engine.unsequa import InputVar

haz_today = haz_base

haz_fut_distr = {
    "x_haz_fut": sp.stats.uniform(1, 3),
}
haz_fut_iv = InputVar(haz_fut_func, haz_fut_distr)

ent_today_distr = {"x_ent": sp.stats.uniform(0.7, 1)}
ent_today_iv = InputVar(ent_today_func, ent_today_distr)

ent_fut_distr = {"m_fut_cost": sp.stats.norm(1, 0.1)}
ent_fut_iv = InputVar(ent_fut_func, ent_fut_distr)
```

```
ent_avg = ent_today_iv.evaluate()
ent_avg.exposures.gdf.head()
```

```
    latitude  longitude         value  deductible         cover  impf_TC  \
0  26.933899 -80.128799  1.671301e+10           0  1.392750e+10        1
1  26.957203 -80.098284  1.511528e+10           0  1.259606e+10        1
2  26.783846 -80.748947  1.511528e+10           0  1.259606e+10        1
3  26.645524 -80.550704  1.511528e+10           0  1.259606e+10        1
4  26.897796 -80.596929  1.511528e+10           0  1.259606e+10        1
```

(continues on next page)

(continued from previous page)

```
     Value_2010
0  5.139301e+09
1  4.647994e+09
2  4.647994e+09
3  4.647994e+09
4  4.647994e+09
```

Compute cost benefit uncertainty and sensitivity using default methods

For examples of how to use non-defaults please see the *impact example*

```
from climada.engine.unsequa import CalcCostBenefit

unc_cb = CalcCostBenefit(
    haz_input_var=haz_today,
    ent_input_var=ent_today_iv,
    haz_fut_input_var=haz_fut_iv,
    ent_fut_input_var=ent_fut_iv,
)
```

```
output_cb = unc_cb.make_sample(N=10, sampling_kwargs={"calc_second_order": False})
output_cb.get_samples_df().tail()
```

```
      x_ent  x_haz_fut  m_fut_cost
45  1.35625    2.96875    0.813727
46  1.04375    2.96875    0.813727
47  1.35625    2.03125    0.813727
48  1.35625    2.96875    0.899001
49  1.04375    2.03125    0.899001
```

For longer computations, it is possible to use a pool for parallel computation.

```
# without pool
output_cb = unc_cb.uncertainty(output_cb)

# with pool
output_cb = unc_cb.uncertainty(output_cb, processes=4)
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           13.8606        10.6498
Beach nourishment          1.71445           10.7904         6.29377
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.4038         3.22132

--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           13.8606        10.6498
Beach nourishment          1.71445           10.7904         6.29377
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.4038         3.22132

--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           14.0781        10.817
Beach nourishment          1.71445           10.968          6.39739
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.5124         3.23322

--------------------  --------  --------
Total climate risk:   118.05    (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3155  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148          14.1012        10.8347
Beach nourishment          1.71445          10.9632         6.39461
Seawall                    8.80916           0.0376243      0.00427104
Building code              9.12786          13.3845         1.46633

--------------------  --------  --------
Total climate risk:   53.5379   (USD bn)
Average annual risk:   6.15933  (USD bn)
Residual risk:        15.0513   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612           13.8606         8.90716
Beach nourishment          2.04988           10.7904         5.2639
Seawall                   10.5327             0.175596       0.0166716
Building code             10.9137            29.4038         2.69421

--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612          14.3188         9.20163
Beach nourishment          2.04988          11.1409         5.4349
Seawall                   10.5327           0.0376243      0.00357216
Building code             10.9137          13.4931         1.23634

--------------------  --------  --------
Total climate risk:   53.9724   (USD bn)
Average annual risk:   6.15933  (USD bn)
Residual risk:        14.982    (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612            6.59956        4.24104
Beach nourishment          2.04988            5.16368        2.51902
Seawall                   10.5327             3.55475        0.337498
Building code             10.9137            48.016          4.3996

--------------------  --------  --------
Total climate risk:   192.064   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.73    (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612            6.43034        4.1323
Beach nourishment          2.04988            5.02552        2.45161
Seawall                   10.5327             3.55475        0.337498
Building code             10.9137            47.9315         4.39186

--------------------  --------  --------
Total climate risk:   191.726   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.784   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612            7.59067         4.87796
Beach nourishment          2.04988            5.96389         2.90939
```

(continues on next page)

(continued from previous page)

```
Seawall                   10.5327              1.31269         0.12463
Building code             10.9137             43.2513          3.96302

--------------------  --------  --------
Total climate risk:   173.005   (USD bn)
Average annual risk:   20.0179  (USD bn)
Residual risk:        114.887   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  -------------------  --------------
Mangroves                  1.30148             6.59956        5.0708
Beach nourishment          1.71445             5.16368        3.01186
Seawall                    8.80916             3.55475        0.403528
Building code              9.12786            48.016          5.26038

--------------------  --------  --------
Total climate risk:   192.064   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.73    (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  -------------------  --------------
Mangroves                  1.30148             7.42146        5.70231
Beach nourishment          1.71445             5.82573        3.39801
Seawall                    8.80916             1.31269        0.149014
Building code              9.12786            43.1668         4.72913

--------------------  --------  --------
Total climate risk:   172.667   (USD bn)
Average annual risk:   20.0179  (USD bn)
Residual risk:        114.941   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  -------------------  --------------
Mangroves                  1.20992           10.5404         8.71168
Beach nourishment          1.59383            8.59532        5.39285
Seawall                    8.18941            0.0184996      0.00225897
Building code              8.48569            7.53759        0.88827

--------------------  --------  --------
Total climate risk:   30.1504   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45852  (USD bn)
--------------------  --------  --------
Net Present Values
```

(continues on next page)

(continued from previous page)

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          10.5646         8.73166
Beach nourishment          1.59383           8.61505        5.40524
Seawall                    8.18941           0.0184996      0.00225897
Building code              8.48569           7.54966        0.889693

--------------------  --------  --------
Total climate risk:   30.1986   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45082  (USD bn)
--------------------  --------  --------
Net Present Values
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992           12.8863        10.6505
Beach nourishment          1.59383            9.98362        6.2639
Seawall                    8.18941            0.257712       0.0314689
Building code              8.48569           33.7244         3.97427

--------------------  --------  --------
Total climate risk:   134.898   (USD bn)
Average annual risk:   15.5605  (USD bn)
Residual risk:         78.0457  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          10.5404         8.71168
Beach nourishment          1.59383           8.59532        5.39285
Seawall                    8.18941           0.0184996      0.00225897
Building code              8.48569           7.53759        0.88827

--------------------  --------  --------
Total climate risk:   30.1504   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45852  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992           12.9104        10.6705
Beach nourishment          1.59383           10.0034         6.27629
Seawall                    8.18941            0.257712       0.0314689
Building code              8.48569           33.7365         3.97569

--------------------  --------  --------
Total climate risk:   134.946   (USD bn)
Average annual risk:   15.5605  (USD bn)
```

(continues on next page)

(continued from previous page)

```
Residual risk:         78.038   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.44955        6.08873
Beach nourishment          1.82807            6.4567         3.53197
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.407          4.25438

--------------------  --------  --------
Total climate risk:   165.628   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.419   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.47373        6.10615
Beach nourishment          1.82807            6.47644        3.54276
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.4191         4.25562

--------------------  --------  --------
Total climate risk:   165.676   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.411   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774         1.39186         1.00297
Beach nourishment          1.82807         1.13491         0.62082
Seawall                    9.39298         0.000227424     2.42121e-05
Building code              9.7328          0.76062         0.0781501

--------------------  ---------  --------
Total climate risk:    3.04248   (USD bn)
Average annual risk:   0.260244  (USD bn)
Residual risk:        -0.245137  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.44955        6.08873
Beach nourishment          1.82807            6.4567         3.53197
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.407          4.25438
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Total climate risk:   165.628   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.419   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774         1.41604         1.02039
Beach nourishment          1.82807         1.15464         0.631618
Seawall                    9.39298         0.000227424     2.42121e-05
Building code              9.7328          0.77269         0.0793903

--------------------  ---------  --------
Total climate risk:    3.09076   (USD bn)
Average annual risk:   0.260244  (USD bn)
Residual risk:        -0.252837  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          2.77518        1.92153
Beach nourishment          1.90253          2.24443        1.17971
Seawall                    9.77553          0.00264385     0.000270456
Building code             10.1292           1.68273        0.166127

--------------------  ---------  --------
Total climate risk:   6.73092    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0259328  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          2.70266        1.87132
Beach nourishment          1.90253          2.18521        1.14859
Seawall                    9.77553          0.00264385     0.000270456
Building code             10.1292           1.64652        0.162552

--------------------  ---------  --------
Total climate risk:   6.58607    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0490347  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
```

(continues on next page)

(continued from previous page)

```
Mangroves                  1.44426          5.43516        3.76329
Beach nourishment          1.90253          4.37024        2.29707
Seawall                    9.77553          0.00764219     0.000781768
Building code             10.1292           3.61055        0.35645

--------------------  --------  --------
Total climate risk:   14.4422   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0186   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          2.77518        1.92153
Beach nourishment          1.90253          2.24443        1.17971
Seawall                    9.77553          0.00264385     0.000270456
Building code             10.1292           1.68273        0.166127

--------------------  ---------  --------
Total climate risk:   6.73092    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0259328  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          5.36264        3.71308
Beach nourishment          1.90253          4.31103        2.26595
Seawall                    9.77553          0.00764219     0.000781768
Building code             10.1292           3.57434        0.352875

--------------------  --------  --------
Total climate risk:   14.2973   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0417   (USD bn)
--------------------  --------  --------
Net Present Values
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259            10.4834         8.32677
Beach nourishment          1.65849           8.23954        4.96809
Seawall                    8.52163           0.415328       0.0487381
Building code              8.82993          36.8908         4.17793

--------------------  --------  --------
Total climate risk:   147.563   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5342  (USD bn)
--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4109         8.26917
Beach nourishment          1.65849            8.18032        4.93239
Seawall                    8.52163            0.415328       0.0487381
Building code              8.82993           36.8546         4.17383

--------------------  --------  --------
Total climate risk:   147.418   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5573  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259              8.79133        6.98278
Beach nourishment          1.65849            6.82168        4.11319
Seawall                    8.52163            0.621468       0.0729283
Building code              8.82993           39.3223         4.45329

--------------------  --------  --------
Total climate risk:   157.289   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.732   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4834         8.32677
Beach nourishment          1.65849            8.23954        4.96809
Seawall                    8.52163            0.415328       0.0487381
Building code              8.82993           36.8908         4.17793

--------------------  --------  --------
Total climate risk:   147.563   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5342  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259              8.71881        6.92518
Beach nourishment          1.65849            6.76247        4.07748
Seawall                    8.52163            0.621468       0.0729283
Building code              8.82993           39.286          4.44919

--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Total climate risk:   157.144   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.755   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288          14.575         10.8535
Beach nourishment          1.76899          11.5436         6.52553
Seawall                    9.08939           0.0620626      0.00682803
Building code              9.41823          19.0306         2.02062

--------------------  --------  --------
Total climate risk:   76.1225   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9112   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288          14.3091        10.6555
Beach nourishment          1.76899          11.3265         6.40279
Seawall                    9.08939           0.0620626      0.00682803
Building code              9.41823          18.8979         2.00652

--------------------  --------  --------
Total climate risk:   75.5914   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9959   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288            7.41063        5.51844
Beach nourishment          1.76899            5.87997        3.32392
Seawall                    9.08939            2.58858        0.284791
Building code              9.41823           46.6706         4.95535

--------------------  --------  --------
Total climate risk:   186.682   (USD bn)
Average annual risk:   21.5984  (USD bn)
Residual risk:        124.133   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888          14.575         12.7977
Beach nourishment          1.50025          11.5436         7.69445
```

(continues on next page)

(continued from previous page)

```
Seawall                    7.70855           0.0620626      0.00805114
Building code              7.98743          19.0306         2.38257

--------------------  --------  --------
Total climate risk:   76.1225   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9112   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ----------------  -------------------  --------------
Mangroves                  1.13888             7.14472        6.27348
Beach nourishment          1.50025             5.66285        3.77461
Seawall                    7.70855             2.58858        0.335806
Building code              7.98743            46.5378         5.82638

--------------------  --------  --------
Total climate risk:   186.151   (USD bn)
Average annual risk:   21.5984  (USD bn)
Residual risk:        124.217   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ----------------  -------------------  --------------
Mangroves                  1.13888             7.63243        6.70172
Beach nourishment          1.50025             5.87772        3.91783
Seawall                    7.70855             1.82863        0.237221
Building code              7.98743            44.9771         5.63099

--------------------  -------  --------
Total climate risk:   179.909  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.593  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ----------------  -------------------  --------------
Mangroves                  1.13888             7.7533         6.80785
Beach nourishment          1.50025             5.97641        3.98361
Seawall                    7.70855             1.82863        0.237221
Building code              7.98743            45.0375         5.63855

--------------------  -------  --------
Total climate risk:   180.15   (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.554  (USD bn)
--------------------  -------  --------
Net Present Values
```

(continues on next page)

(continued from previous page)

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888           14.9037        13.0863
Beach nourishment          1.50025           11.752          7.83335
Seawall                    7.70855            0.108627       0.0140917
Building code              7.98743           24.3925         3.05386

--------------------  -------  --------
Total climate risk:   97.5699  (USD bn)
Average annual risk:  11.2725  (USD bn)
Residual risk:        46.4132  (USD bn)
--------------------  -------  --------
Net Present Values
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288             7.63243        5.68361
Beach nourishment          1.76899             5.87772        3.32264
Seawall                    9.08939             1.82863        0.201183
Building code              9.41823            44.9771         4.77554

--------------------  -------  --------
Total climate risk:   179.909  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.593  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288           15.0245        11.1883
Beach nourishment          1.76899           11.8507         6.69911
Seawall                    9.08939            0.108627       0.0119509
Building code              9.41823           24.4528         2.59633

--------------------  -------  --------
Total climate risk:   97.8114  (USD bn)
Average annual risk:  11.2725  (USD bn)
Residual risk:        46.3747  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205          5.31429        4.01972
Beach nourishment          1.74155          4.27155        2.45273
Seawall                    8.9484           0.00764219     0.000854029
Building code              9.27214          3.5502         0.382889

--------------------  --------  --------
Total climate risk:   14.2008   (USD bn)
Average annual risk:   1.57569  (USD bn)
```

(continues on next page)

(continued from previous page)

```
Residual risk:         1.0571   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205          5.5802         4.22086
Beach nourishment          1.74155          4.48867        2.5774
Seawall                    8.9484           0.00764219     0.000854029
Building code              9.27214          3.68297        0.397208

--------------------  ---------  --------
Total climate risk:   14.7319    (USD bn)
Average annual risk:   1.57569   (USD bn)
Residual risk:         0.972395  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205             7.60825        5.75487
Beach nourishment          1.74155             5.85798        3.36366
Seawall                    8.9484              1.82863        0.204353
Building code              9.27214            44.9651         4.84948

--------------------  -------  --------
Total climate risk:   179.86   (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.6    (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.36453          5.31429        3.89458
Beach nourishment          1.79751          4.27155        2.37637
Seawall                    9.23593          0.00764219     0.000827442
Building code              9.57007          3.5502         0.370969

--------------------  --------  --------
Total climate risk:   14.2008   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0571   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.36453             7.87416        5.77059
Beach nourishment          1.79751             6.0751         3.37973
Seawall                    9.23593             1.82863        0.197991
Building code              9.57007            45.0978         4.71238
```

(continues on next page)

(continued from previous page)

```
--------------------  -------  --------
Total climate risk:   180.391  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.516  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742            8.67047        8.12282
Beach nourishment          1.40612            6.72299        4.78124
Seawall                    7.2249             0.621468       0.0860176
Building code              7.48629           39.2619         5.24451

--------------------  --------  --------
Total climate risk:   157.048   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.771   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742            8.5496         8.00959
Beach nourishment          1.40612            6.6243         4.71105
Seawall                    7.2249             0.621468       0.0860176
Building code              7.48629           39.2016         5.23645

--------------------  --------  --------
Total climate risk:   156.806   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.809   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742           14.5509        13.6318
Beach nourishment          1.40612           11.5238         8.19549
Seawall                    7.2249             0.0620626      0.0085901
Building code              7.48629           19.0186         2.54045

--------------------  --------  --------
Total climate risk:   76.0742   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9189   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
```

(continues on next page)

(continued from previous page)

```
Mangroves                  1.17928            8.67047        7.35233
Beach nourishment          1.55347            6.72299        4.32772
Seawall                    7.98203            0.621468       0.0778584
Building code              8.27081           39.2619         4.74705

--------------------  --------  --------
Total climate risk:   157.048   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.771   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.17928          14.43          12.2363
Beach nourishment          1.55347          11.4252         7.35459
Seawall                    7.98203           0.0620626      0.00777529
Building code              8.27081          18.9582         2.29218

--------------------  --------  --------
Total climate risk:   75.8328   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9574   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           13.8606        10.6498
Beach nourishment          1.71445           10.7904         6.29377
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.4038         3.22132

--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
--------------------  --------  --------
Net Present Values
```

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           13.8606        10.6498
Beach nourishment          1.71445           10.7904         6.29377
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.4038         3.22132


--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Measure               Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          10.5404         8.71168
Beach nourishment          1.59383           8.59532        5.39285
Seawall                    8.18941           0.0184996      0.00225897
Building code              8.48569           7.53759        0.88827Net Present Values


--------------------  --------  --------
Total climate risk:   30.1504   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45852  (USD bn)
--------------------  --------  --------
Net Present Values

Measure               Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4109         8.26917
Beach nourishment          1.65849            8.18032        4.93239
Seawall                    8.52163            0.415328       0.0487381
Building code              8.82993           36.8546         4.17383

--------------------  --------  --------
Total climate risk:   147.418   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5573  (USD bn)
--------------------  --------  --------
Net Present Values

Measure               Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288           15.0245        11.1883
Beach nourishment          1.76899           11.8507         6.69911
Seawall                    9.08939            0.108627       0.0119509
Building code              9.41823           24.4528         2.59633

--------------------  -------  --------
Total climate risk:   97.8114  (USD bn)
Average annual risk:  11.2725  (USD bn)
Residual risk:        46.3747  (USD bn)
--------------------  -------  --------
Net Present Values

Measure               Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148           14.0781        10.817
Beach nourishment          1.71445           10.968          6.39739
Seawall                    8.80916            0.175596       0.0199334
Building code              9.12786           29.5124         3.23322

--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Total climate risk:   118.05    (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3155  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992           12.9104        10.6705
Beach nourishment          1.59383           10.0034         6.27629
Seawall                    8.18941            0.257712       0.0314689
Building code              8.48569           33.7365         3.97569

--------------------  --------  --------
Total climate risk:   134.946   (USD bn)
Average annual risk:   15.5605  (USD bn)
Residual risk:         78.038   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259              8.79133        6.98278
Beach nourishment          1.65849            6.82168        4.11319
Seawall                    8.52163            0.621468       0.0729283
Building code              8.82993           39.3223         4.45329


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205          5.31429        4.01972
Beach nourishment          1.74155          4.27155        2.45273
Seawall                    8.9484           0.00764219     0.000854029
Building code              9.27214          3.5502         0.382889-------------------
↪-  --------  --------
Total climate risk:   157.289   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.732   (USD bn)
--------------------  --------  --------


Net Present Values--------------------  --------  --------
Total climate risk:   14.2008   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0571   (USD bn)
--------------------  --------  --------

Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148          14.1012        10.8347
```

(continues on next page)

(continued from previous page)

```
Beach nourishment          1.71445          10.9632         6.39461
Seawall                    8.80916           0.0376243      0.00427104
Building code              9.12786          13.3845         1.46633

--------------------  --------  --------
Total climate risk:   53.5379   (USD bn)
Average annual risk:   6.15933  (USD bn)
Residual risk:        15.0513   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.44955        6.08873
Beach nourishment          1.82807            6.4567         3.53197
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.407          4.25438

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4834         8.32677
Beach nourishment          1.65849            8.23954        4.96809
Seawall                    8.52163            0.415328       0.0487381
Building code              8.82993           36.8908         4.17793


--------------------  --------  --------
Total climate risk:   165.628   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.419   (USD bn)
--------------------  --------  --------
--------------------  --------  --------
Total climate risk:   147.563   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5342  (USD bn)
--------------------  --------  --------Net Present Values

Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205          5.5802         4.22086
Beach nourishment          1.74155          4.48867        2.5774
Seawall                    8.9484           0.00764219     0.000854029
Building code              9.27214          3.68297        0.397208

--------------------  ---------  --------
Total climate risk:   14.7319    (USD bn)
Average annual risk:   1.57569   (USD bn)
Residual risk:         0.972395  (USD bn)
--------------------  ---------  --------
Net Present Values
```

(continues on next page)

(continued from previous page)

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612           13.8606         8.90716
Beach nourishment          2.04988           10.7904         5.2639
Seawall                   10.5327             0.175596       0.0166716
Building code             10.9137            29.4038         2.69421

--------------------  --------  --------
Total climate risk:   117.615   (USD bn)
Average annual risk:   13.6166  (USD bn)
Residual risk:         63.3848  (USD bn)
```

```
--------------------  --------  --------
Net Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.47373        6.10615
Beach nourishment          1.82807            6.47644        3.54276
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.4191         4.25562Measure
↪   Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259              8.71881        6.92518
Beach nourishment          1.65849            6.76247        4.07748
Seawall                    8.52163            0.621468       0.0729283
Building code              8.82993           39.286          4.44919



--------------------  --------  --------
Total climate risk:   165.676   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.411   (USD bn)
--------------------  --------  ----------------------------  --------  --------
Total climate risk:   157.144   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.755   (USD bn)
--------------------  --------  --------

Net Present ValuesNet Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.32205             7.60825        5.75487
Beach nourishment          1.74155             5.85798        3.36366
Seawall                    8.9484              1.82863        0.204353
Building code              9.27214            44.9651         4.84948
```

(continues on next page)

(continued from previous page)

```
--------------------  -------  --------
Total climate risk:   179.86   (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.6    (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612          14.3188         9.20163
Beach nourishment          2.04988          11.1409         5.4349
Seawall                   10.5327           0.0376243      0.00357216
Building code             10.9137          13.4931         1.23634

--------------------  --------  --------
Total climate risk:   53.9724   (USD bn)
Average annual risk:   6.15933  (USD bn)
Residual risk:        14.982    (USD bn)
--------------------  --------  --------

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774         1.39186         1.00297
Beach nourishment          1.82807         1.13491         0.62082
Seawall                    9.39298         0.000227424     2.42121e-05
Building code              9.7328          0.76062         0.0781501Net Present Values


--------------------  ---------  --------
Total climate risk:    3.04248   (USD bn)
Average annual risk:   0.260244  (USD bn)
Residual risk:        -0.245137  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288          14.575         10.8535
Beach nourishment          1.76899          11.5436         6.52553
Seawall                    9.08939           0.0620626      0.00682803
Building code              9.41823          19.0306         2.02062

--------------------  --------  --------
Total climate risk:   76.1225   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9112   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.36453          5.31429        3.89458
```

(continues on next page)

(continued from previous page)

```
Beach nourishment          1.79751          4.27155        2.37637
Seawall                    9.23593          0.00764219     0.000827442
Building code              9.57007          3.5502         0.370969

--------------------  --------  --------
Total climate risk:   14.2008   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0571   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774            8.44955        6.08873
Beach nourishment          1.82807            6.4567         3.53197
Seawall                    9.39298            0.895618       0.0953497
Building code              9.7328            41.407          4.25438

--------------------  --------  --------
Total climate risk:   165.628   (USD bn)
Average annual risk:   19.1818  (USD bn)
Residual risk:        108.419   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612             6.59956        4.24104
Beach nourishment          2.04988             5.16368        2.51902
Seawall                   10.5327              3.55475        0.337498
Building code             10.9137             48.016          4.3996

--------------------  --------  --------
Total climate risk:   192.064   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.73    (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288          14.3091        10.6555
Beach nourishment          1.76899          11.3265         6.40279
Seawall                    9.08939           0.0620626      0.00682803
Building code              9.41823          18.8979         2.00652


--------------------  --------  --------
Total climate risk:   75.5914   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9959   (USD bn)
--------------------  --------  --------
```

(continues on next page)

(continued from previous page)

```
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.36453             7.87416        5.77059
Beach nourishment          1.79751             6.0751         3.37973
Seawall                    9.23593             1.82863        0.197991
Building code              9.57007            45.0978         4.71238
Net Present Values

--------------------  -------  --------
Total climate risk:   180.391  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.516  (USD bn)
--------------------  -------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.38774         1.41604         1.02039
Beach nourishment          1.82807         1.15464         0.631618
Seawall                    9.39298         0.000227424     2.42121e-05
Building code              9.7328          0.77269         0.0793903

--------------------  ---------  --------
Total climate risk:    3.09076   (USD bn)
Average annual risk:   0.260244  (USD bn)
Residual risk:        -0.252837  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612             6.43034        4.1323
Beach nourishment          2.04988             5.02552        2.45161
Seawall                   10.5327              3.55475        0.337498
Building code             10.9137             47.9315         4.39186

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288             7.41063        5.51844
Beach nourishment          1.76899             5.87997        3.32392
Seawall                    9.08939             2.58858        0.284791
Building code              9.41823            46.6706         4.95535


--------------------  --------  --------
Total climate risk:   191.726   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.784   (USD bn)
--------------------  --------  ----------------------------  --------  --------
Total climate risk:   186.682   (USD bn)
Average annual risk:   21.5984  (USD bn)
Residual risk:        124.133   (USD bn)
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Net Present Values

Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742            8.67047        8.12282
Beach nourishment          1.40612            6.72299        4.78124
Seawall                    7.2249             0.621468       0.0860176
Building code              7.48629           39.2619         5.24451

--------------------  --------  --------
Total climate risk:   157.048   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.771   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          2.77518        1.92153
Beach nourishment          1.90253          2.24443        1.17971
Seawall                    9.77553          0.00264385     0.000270456
Building code             10.1292           1.68273        0.166127

--------------------  ---------  --------
Total climate risk:   6.73092    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0259328  (USD bn)
--------------------  ---------  --------
Net Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.55612             7.59067         4.87796
Beach nourishment          2.04988             5.96389         2.90939
Seawall                   10.5327              1.31269         0.12463
Building code             10.9137             43.2513          3.96302Measure
↪     Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888          14.575         12.7977
Beach nourishment          1.50025          11.5436         7.69445
Seawall                    7.70855           0.0620626      0.00805114
Building code              7.98743          19.0306         2.38257



--------------------  --------  --------
Total climate risk:   173.005   (USD bn)
Average annual risk:   20.0179  (USD bn)
```

(continues on next page)

(continued from previous page)

```
Residual risk:        114.887   (USD bn)
--------------------  --------  ----------------------------  --------  --------
Total climate risk:   76.1225   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9112   (USD bn)
--------------------  --------  --------

Net Present ValuesNet Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742            8.5496         8.00959
Beach nourishment          1.40612            6.6243         4.71105
Seawall                    7.2249             0.621468       0.0860176
Building code              7.48629           39.2016         5.23645

--------------------  --------  --------
Total climate risk:   156.806   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.809   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          2.70266        1.87132
Beach nourishment          1.90253          2.18521        1.14859
Seawall                    9.77553          0.00264385     0.000270456
Building code             10.1292           1.64652        0.162552


--------------------  ---------  --------
Total climate risk:   6.58607    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0490347  (USD bn)
--------------------  ---------  --------Measure              Cost (USD bn)    ␣
↪Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148            6.59956        5.0708
Beach nourishment          1.71445            5.16368        3.01186
Seawall                    8.80916            3.55475        0.403528
Building code              9.12786           48.016          5.26038

Net Present Values
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888            7.14472        6.27348
Beach nourishment          1.50025            5.66285        3.77461
Seawall                    7.70855            2.58858        0.335806
Building code              7.98743           46.5378         5.82638
```

(continues on next page)

(continued from previous page)

```


--------------------  --------  --------
Total climate risk:   192.064   (USD bn)
Average annual risk:   22.2359  (USD bn)
Residual risk:        128.73    (USD bn)
--------------------  --------  --------
--------------------  --------  --------
Total climate risk:   186.151   (USD bn)
Average annual risk:   21.5984  (USD bn)
Residual risk:        124.217   (USD bn)
--------------------  --------  --------Net Present Values
Net Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.06742          14.5509         13.6318
Beach nourishment          1.40612          11.5238          8.19549
Seawall                    7.2249            0.0620626       0.0085901
Building code              7.48629          19.0186          2.54045

--------------------  --------  --------
Total climate risk:   76.0742   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9189   (USD bn)
--------------------  --------  --------
Net Present Values


Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888             7.63243        6.70172
Beach nourishment          1.50025             5.87772        3.91783
Seawall                    7.70855             1.82863        0.237221
Building code              7.98743            44.9771         5.63099Measure
↪    Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.30148             7.42146        5.70231
Beach nourishment          1.71445             5.82573        3.39801
Seawall                    8.80916             1.31269        0.149014
Building code              9.12786            43.1668         4.72913



Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          5.43516        3.76329
Beach nourishment          1.90253          4.37024        2.29707
Seawall                    9.77553          0.00764219     0.000781768
Building code             10.1292           3.61055        0.35645
--------------------  -------  --------
```

(continues on next page)

(continued from previous page)

```
Total climate risk:   179.909  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.593  (USD bn)
```

```
--------------------  -------  --------
--------------------  --------  --------
Total climate risk:   172.667   (USD bn)
Average annual risk:   20.0179  (USD bn)
Residual risk:        114.941   (USD bn)
--------------------  --------  --------

Net Present Values

--------------------  --------  --------
Total climate risk:   14.4422   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0186   (USD bn)
--------------------  --------  --------Net Present Values

Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.17928            8.67047        7.35233
Beach nourishment          1.55347            6.72299        4.32772
Seawall                    7.98203            0.621468       0.0778584
Building code              8.27081           39.2619         4.74705

--------------------  --------  --------
Total climate risk:   157.048   (USD bn)
Average annual risk:   18.1551  (USD bn)
Residual risk:        101.771   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888             7.7533         6.80785
Beach nourishment          1.50025             5.97641        3.98361
Seawall                    7.70855             1.82863        0.237221
Building code              7.98743            45.0375         5.63855

--------------------  -------  --------
Total climate risk:   180.15   (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.554  (USD bn)
--------------------  -------  --------

Net Present ValuesMeasure              Cost (USD bn)    Benefit (USD bn)    Benefit/
↪Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          10.5404         8.71168
```

(continues on next page)

(continued from previous page)

```
Beach nourishment          1.59383           8.59532        5.39285
Seawall                    8.18941           0.0184996      0.00225897
Building code              8.48569           7.53759        0.88827



--------------------  --------  --------
Total climate risk:   30.1504   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45852  (USD bn)
--------------------  --------  --------
Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          2.77518        1.92153
Beach nourishment          1.90253          2.24443        1.17971
Seawall                    9.77553          0.00264385     0.000270456
Building code             10.1292           1.68273        0.166127Net Present Values


--------------------  ---------  --------
Total climate risk:   6.73092    (USD bn)
Average annual risk:  0.678263   (USD bn)
Residual risk:        0.0259328  (USD bn)
--------------------  ---------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.17928          14.43          12.2363
Beach nourishment          1.55347          11.4252         7.35459
Seawall                    7.98203           0.0620626      0.00777529
Building code              8.27081          18.9582         2.29218

--------------------  --------  --------
Total climate risk:   75.8328   (USD bn)
Average annual risk:   8.73152  (USD bn)
Residual risk:        30.9574   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.13888           14.9037        13.0863
Beach nourishment          1.50025           11.752          7.83335
Seawall                    7.70855            0.108627       0.0140917
Building code              7.98743           24.3925         3.05386


--------------------  -------  --------
Total climate risk:   97.5699  (USD bn)
Average annual risk:  11.2725  (USD bn)
Residual risk:        46.4132  (USD bn)
```

(continues on next page)

(continued from previous page)

```
--------------------  -------  --------Measure              Cost (USD bn)    Benefit␣
↪(USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          10.5646         8.73166
Beach nourishment          1.59383           8.61505        5.40524
Seawall                    8.18941           0.0184996      0.00225897
Building code              8.48569           7.54966        0.889693

Net Present Values

--------------------  --------  --------
Total climate risk:   30.1986   (USD bn)
Average annual risk:   3.37008  (USD bn)
Residual risk:         3.45082  (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.44426          5.36264        3.71308
Beach nourishment          1.90253          4.31103        2.26595
Seawall                    9.77553          0.00764219     0.000781768
Building code             10.1292           3.57434        0.352875

--------------------  --------  --------
Total climate risk:   14.2973   (USD bn)
Average annual risk:   1.57569  (USD bn)
Residual risk:         1.0417   (USD bn)
--------------------  --------  --------
Net Present Values

Measure              Cost (USD bn)    Benefit (USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.34288            7.63243        5.68361
Beach nourishment          1.76899            5.87772        3.32264
Seawall                    9.08939            1.82863        0.201183
Building code              9.41823           44.9771         4.77554


--------------------  -------  --------
Total climate risk:   179.909  (USD bn)
Average annual risk:   20.855  (USD bn)
Residual risk:        119.593  (USD bn)
--------------------  -------  --------Measure              Cost (USD bn)    Benefit␣
↪(USD bn)    Benefit/Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.20992          12.8863        10.6505
Beach nourishment          1.59383           9.98362        6.2639
Seawall                    8.18941           0.257712       0.0314689
Building code              8.48569          33.7244         3.97427

Net Present Values
```

(continues on next page)

(continued from previous page)

```
--------------------  --------  --------
Total climate risk:   134.898   (USD bn)
Average annual risk:   15.5605  (USD bn)
Residual risk:         78.0457  (USD bn)
--------------------  --------  --------

Net Present ValuesMeasure              Cost (USD bn)    Benefit (USD bn)    Benefit/
↪Cost
-----------------  ---------------  ------------------  --------------
Mangroves                  1.259             10.4834         8.32677
Beach nourishment          1.65849            8.23954        4.96809
Seawall                    8.52163            0.415328       0.0487381
Building code              8.82993           36.8908         4.17793


--------------------  --------  --------
Total climate risk:   147.563   (USD bn)
Average annual risk:   17.0232  (USD bn)
Residual risk:         91.5342  (USD bn)
--------------------  --------  --------
Net Present Values
```

The output of `CostBenefit.calc` is rather complex in its structure. The metrics dictionary inherits this complexity.

```
# Top level metrics keys
macro_metrics = output_cb.uncertainty_metrics
macro_metrics
```

```
['imp_meas_present',
 'imp_meas_future',
 'tot_climate_risk',
 'benefit',
 'cost_ben_ratio']
```

```
# The benefits and cost_ben_ratio are available for each measure
output_cb.get_uncertainty(metric_list=["benefit", "cost_ben_ratio"]).tail()
```

```
    Mangroves Benef  Beach nourishment Benef  Seawall Benef  \
45     8.670468e+09             6.722992e+09   6.214684e+08
46     8.549601e+09             6.624301e+09   6.214684e+08
47     1.455086e+10             1.152385e+10   6.206260e+07
48     8.670468e+09             6.722992e+09   6.214684e+08
49     1.443000e+10             1.142516e+10   6.206260e+07

    Building code Benef  Mangroves CostBen  Beach nourishment CostBen  \
45         3.926190e+10           0.123110                   0.209151
46         3.920155e+10           0.124850                   0.212267
47         1.901856e+10           0.073358                   0.122018
48         3.926190e+10           0.136011                   0.231069
49         1.895821e+10           0.081724                   0.135970
```

(continues on next page)

(continued from previous page)

```
    Seawall CostBen  Building code CostBen
45        11.625533               0.190676
46        11.625533               0.190969
47       116.413127               0.393631
48        12.843826               0.210657
49       128.612593               0.436265
```

```
# The impact_meas_present and impact_meas_future provide values of the cost_meas,␣
↪risk_transf, risk,
# and cost_ins for each measure
output_cb.get_uncertainty(metric_list=["imp_meas_present"]).tail()
```

```
    no measure - risk - present  no measure - risk_transf - present  \
45                 1.040893e+08                                 0.0
46                 8.010560e+07                                 0.0
47                 1.040893e+08                                 0.0
48                 1.040893e+08                                 0.0
49                 8.010560e+07                                 0.0

    no measure - cost_meas - present  no measure - cost_ins - present  \
45                                 0                                 0
46                                 0                                 0
47                                 0                                 0
48                                 0                                 0
49                                 0                                 0

    Mangroves - risk - present  Mangroves - risk_transf - present  \
45                5.197409e+07                                 0
46                3.999849e+07                                 0
47                5.197409e+07                                 0
48                5.197409e+07                                 0
49                3.999849e+07                                 0

    Mangroves - cost_meas - present  Mangroves - cost_ins - present  \
45                     1.311768e+09                                1
46                     1.311768e+09                                1
47                     1.311768e+09                                1
48                     1.311768e+09                                1
49                     1.311768e+09                                1

    Beach nourishment - risk - present  \
45                        6.153578e+07
46                        4.735703e+07
47                        6.153578e+07
48                        6.153578e+07
49                        4.735703e+07

    Beach nourishment - risk_transf - present  \
45                                          0
46                                          0
```

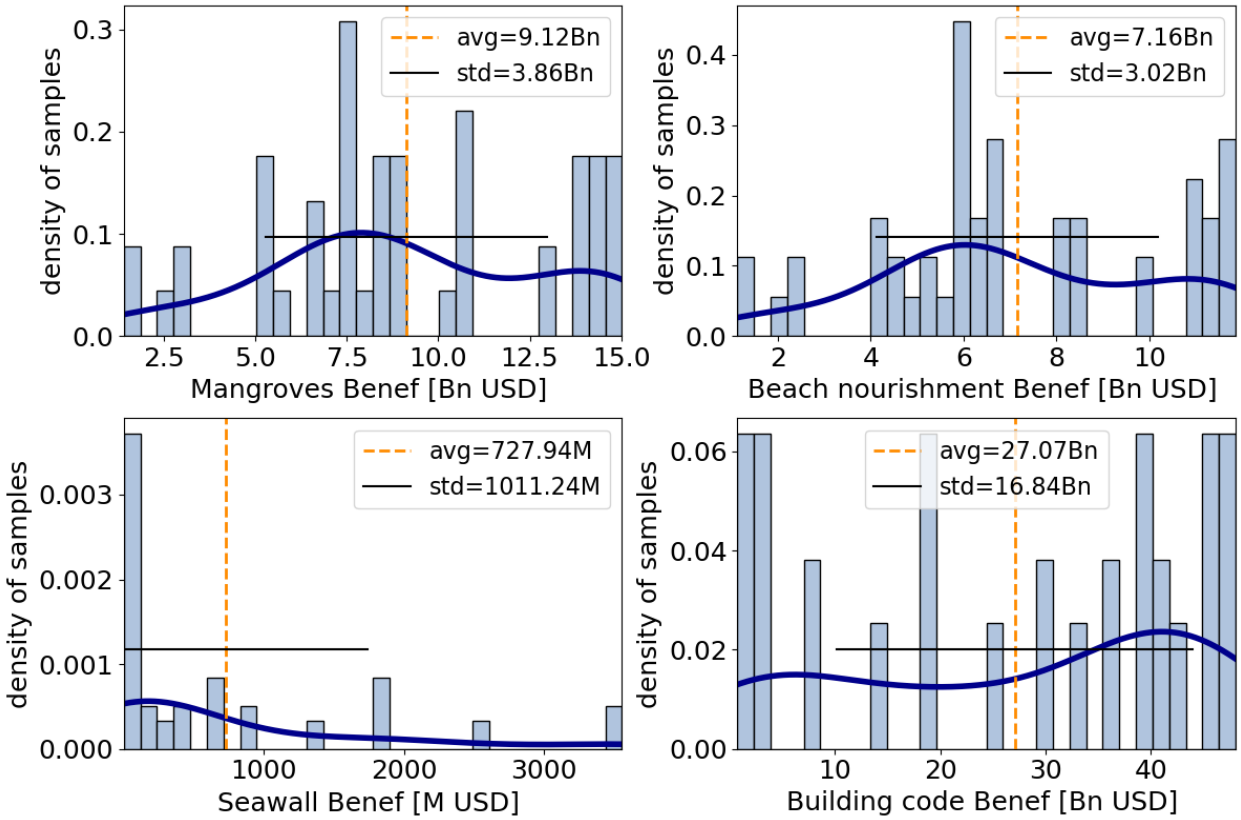
(continues on next page)

(continued from previous page)

```
47                                          0
48                                          0
49                                          0

    Beach nourishment - cost_meas - present  \
45                            1.728000e+09
46                            1.728000e+09
47                            1.728000e+09
48                            1.728000e+09
49                            1.728000e+09

    Beach nourishment - cost_ins - present  Seawall - risk - present  \
45                                       1              1.040893e+08
46                                       1              8.010560e+07
47                                       1              1.040893e+08
48                                       1              1.040893e+08
49                                       1              8.010560e+07

    Seawall - risk_transf - present  Seawall - cost_meas - present  \
45                               0                   8.878779e+09
46                               0                   8.878779e+09
47                               0                   8.878779e+09
48                               0                   8.878779e+09
49                               0                   8.878779e+09

    Seawall - cost_ins - present  Building code - risk - present  \
45                            1                    7.806698e+07
46                            1                    6.007920e+07
47                            1                    7.806698e+07
48                            1                    7.806698e+07
49                            1                    6.007920e+07

    Building code - risk_transf - present  \
45                                      0
46                                      0
47                                      0
48                                      0
49                                      0

    Building code - cost_meas - present  Building code - cost_ins - present
45                        9.200000e+09                                    1
46                        9.200000e+09                                    1
47                        9.200000e+09                                    1
48                        9.200000e+09                                    1
49                        9.200000e+09                                    1
```

We can plot the distributions for the top metrics or our choice.

```
# tot_climate_risk and benefit
output_cb.plot_uncertainty(metric_list=["benefit"], figsize=(12, 8));
```



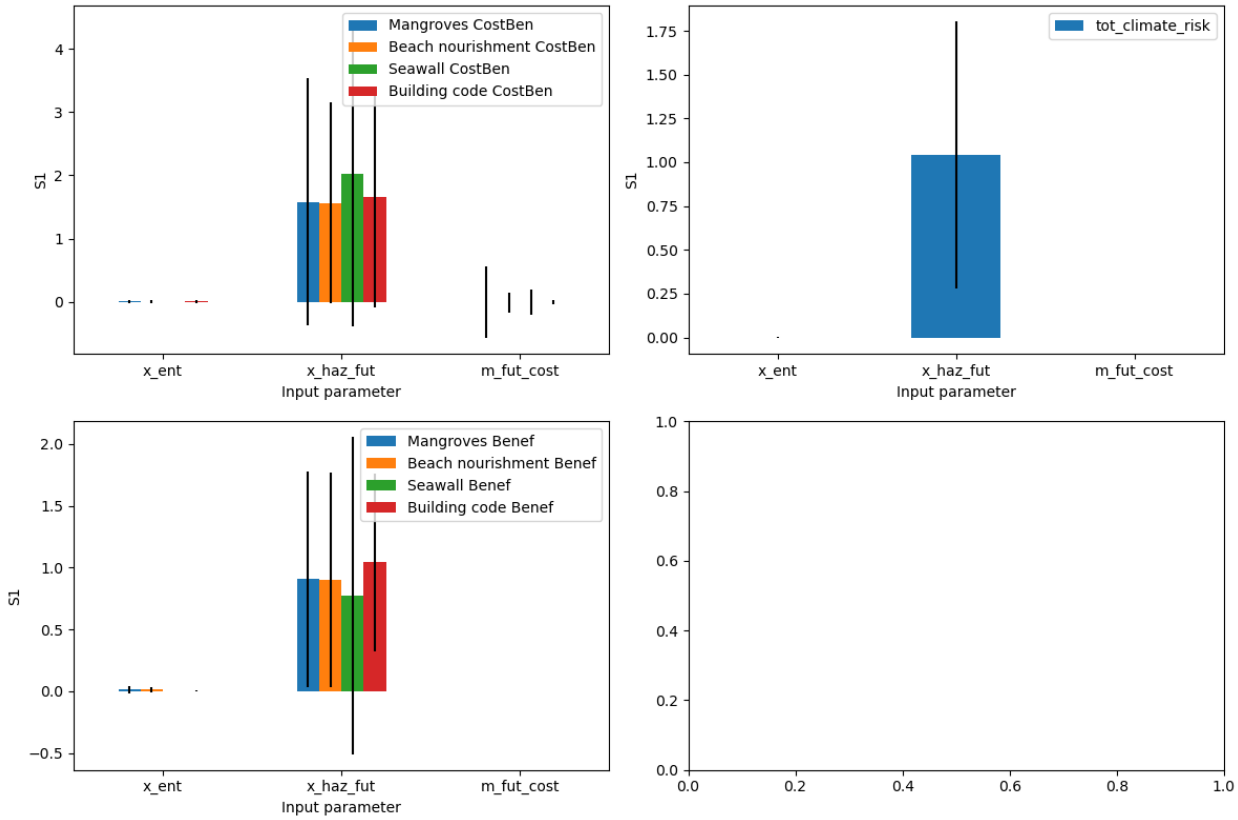
Analogously to the impact example, now that we have a metric distribution, we can compute the sensitivity indices. Since we used the default sampling method, we can use the default sensitivity analysis method. However, since we used `calc_second_order = False` for the sampling, we need to specify the same for the sensitivity analysis.

```
output_cb = unc_cb.sensitivity(
    output_cb, sensitivity_kwargs={"calc_second_order": False}
)
```

The sensitivity indices can be plotted. For the default method 'sobol', by default the 'S1' sensitivity index is plotted.

Note that since we have quite a few measures, the plot must be adjusted a bit or dropped. Also see that for many metrics, the sensitivity to certain uncertainty parameters appears to be 0. However, this result is to be treated with care. Indeed, we used for demonstration purposes a rather too low number of samples, which is indicated by large confidence intervals (vertical black lines) for most sensitivity indices. For a more robust result the analysis should be repeated with more samples.

```
# plot only certain metrics
axes = output_cb.plot_sensitivity(
    metric_list=["cost_ben_ratio", "tot_climate_risk", "benefit"], figsize=(12, 8)
);
```



Advanced examples

Coupled variables

In this example, we show how you can define correlated input variables. Suppose your exposures and hazards are conditioned on the same Shared Socio-economic Pathway (SSP). Then, you want that only exposures and hazard belonging to the same SSP are present in each sample.

In order to achieve this, you must simply define an uncertainty parameter that shares the same name and the same distribution for both the exposures and the hazard uncertainty variables.

Many scenarios of hazards and exposures

In this example we look at the case where many scenarios are tested in the uncertainty analysis. For instance, suppose you have data for different Shared Socio-economic Pathways (SSP) and different Climate Change projections. From the SSPs, you have a number of Exposures, saved to files. From the climate projections, you have a number of Hazards, saved to file.

The task is to sample from the SSPs and the Climate change scenarios for the uncertainty and sensitivity analysis efficiently.

For demonstration purposes, we will use below as exposures files the litpop for three countries, and for tha hazard files the winter storms for the same three countries. Instead of having SSPs, we now want to only combine exposures and hazards of the same countries.

```python
from climada.util.api_client import Client

client = Client()
```

```python
def get_litpop(iso):
    return client.get_litpop(country=iso)


def get_ws(iso):
    properties = {
        "country_iso3alpha": iso,
    }
    return client.get_hazard("storm_europe", properties=properties)
```

```python
# Define list of exposures and/or of hazard files

exp_list = [get_litpop(iso) for iso in ["CHE", "DEU", "ITA"]]
haz_list = [get_ws(iso) for iso in ["CHE", "DEU", "ITA"]]
for exp, haz in zip(exp_list, haz_list):
    exp.gdf["impf_WS"] = 1
    exp.assign_centroids(haz)
```

```python
# Define the input variable
from climada.entity import ImpactFuncSet, Exposures
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.hazard import Hazard
from climada.engine.unsequa import InputVar
import scipy as sp
import copy


def exp_func(cnt, x_exp, exp_list=exp_list):
    exp = exp_list[int(cnt)].copy()
    exp.gdf["value"] *= x_exp
    return exp


exp_distr = {
    "x_exp": sp.stats.uniform(0.9, 0.2),
    "cnt": sp.stats.randint(
        low=0, high=len(exp_list)
    ),  # use the same parameter name accross input variables
}
exp_iv = InputVar(exp_func, exp_distr)


def haz_func(cnt, i_haz, haz_list=haz_list):
    haz = copy.deepcopy(
        haz_list[int(cnt)]
    )  # use the same parameter name accross input variables
    haz.intensity *= i_haz
    return haz
```

(continues on next page)

(continued from previous page)

```
haz_distr = {
    "i_haz": sp.stats.norm(1, 0.2),
    "cnt": sp.stats.randint(low=0, high=len(haz_list)),
}
haz_iv = InputVar(haz_func, haz_distr)

impf = ImpfStormEurope.from_schwierz()
impf_set = ImpactFuncSet()
impf_set.append(impf)
impf_iv = InputVar.impfset([impf_set], bounds_mdd=[0.9, 1.1])
```

```
from climada.engine.unsequa import CalcImpact

calc_imp = CalcImpact(exp_iv, impf_iv, haz_iv)
```

```
2024-01-25 15:38:30,713 - climada.engine.unsequa.calc_base - WARNING -

The input parameter cnt is shared among at least 2 input variables. Their uncertainty␣
↪is thus computed with the same samples for this input paramter.
```

```
output_imp = calc_imp.make_sample(N=2**2, sampling_kwargs={"skip_values": 2**3})
```

```
# as we can see, there is only a single input parameter "cnt" to select the country␣
↪for both the exposures and the hazard
output_imp.samples_df.tail()
```

```
     x_exp  cnt     MDD     i_haz
35  0.9875  0.0  1.0375  1.097755
36  1.0625  1.0  1.0375  1.097755
37  1.0625  0.0  0.9375  1.097755
38  1.0625  0.0  1.0375  1.097755
39  1.0625  0.0  1.0375  1.097755
```

```
output_imp = calc_imp.uncertainty(output_imp)
```

```
output_imp.aai_agg_unc_df.tail()
```

Input variable: Repeated loading of files made efficient

Loading Hazards or Exposures from file is a rather lengthy operation. Thus, we want to minimize the reading operations, ideally reading each file only once. Simultaneously, Hazard and Exposures can be large in memory, and thus we would like to have at most one of each loaded at a time. Thus, we do not want to use the list capacity from the helper method InputVar.exposures and InputVar.hazard.

For demonstration purposes, we will use below as exposures files the litpop for three countries, and for tha hazard files the winter storms for the same three countries. Note that this does not make a lot of sense for an uncertainty analysis. For your use case, please replace the set of exposures and/or hazard files with meaningful sets, for instance sets of exposures for different resolutions or hazards for different model runs.

```
from climada.util.api_client import Client

client = Client()
```

```
def get_litpop_path(iso):
    properties = {
        "country_iso3alpha": iso,
        "res_arcsec": "150",
        "exponents": "(1,1)",
        "fin_mode": "pc",
    }
    litpop_datasets = client.list_dataset_infos(
        data_type="litpop", properties=properties
    )
    ds = litpop_datasets[0]
    download_dir, ds_files = client.download_dataset(ds)
    return ds_files[0]


def get_ws_path(iso):
    properties = {
        "country_iso3alpha": iso,
    }
    hazard_datasets = client.list_dataset_infos(
        data_type="storm_europe", properties=properties
    )
    ds = hazard_datasets[0]
    download_dir, ds_files = client.download_dataset(ds)
    return ds_files[0]
```

```
# Define list of exposures and/or of hazard files

f_exp_list = [get_litpop_path(iso) for iso in ["CHE", "DEU", "ITA"]]
f_haz_list = [get_ws_path(iso) for iso in ["CHE", "DEU", "ITA"]]
```

```
# Define the input variable for the loading files
# The trick is to not reload a file if it is already in memory. This is done using a␣
↪global variable.
from climada.entity import ImpactFunc, ImpactFuncSet, Exposures
from climada.hazard import Hazard
from climada.engine.unsequa import InputVar
import scipy as sp
import copy


def exp_func(f_exp, x_exp, filename_list=f_exp_list):
    filename = filename_list[int(f_exp)]
    global exp_base
    if "exp_base" in globals():
        if isinstance(exp_base, Exposures):
            if exp_base.gdf["filename"] != str(filename):
                exp_base = Exposures.from_hdf5(filename)
```

(continues on next page)

(continued from previous page)

```
                exp_base.gdf["filename"] = str(filename)
    else:
        exp_base = Exposures.from_hdf5(filename)
        exp_base.gdf["filename"] = str(filename)

    exp = exp_base.copy()
    exp.gdf["value"] *= x_exp
    return exp


exp_distr = {
    "x_exp": sp.stats.uniform(0.9, 0.2),
    "f_exp": sp.stats.randint(low=0, high=len(f_exp_list)),
}
exp_iv = InputVar(exp_func, exp_distr)


def haz_func(f_haz, i_haz, filename_list=f_haz_list):
    filename = filename_list[int(f_haz)]
    global haz_base
    if "haz_base" in globals():
        if isinstance(haz_base, Hazard):
            if haz_base.filename != str(filename):
                haz_base = Hazard.from_hdf5(filename)
                haz_base.filename = str(filename)
    else:
        haz_base = Hazard.from_hdf5(filename)
        haz_base.filename = str(filename)

    haz = copy.deepcopy(haz_base)
    haz.intensity *= i_haz
    return haz


haz_distr = {
    "i_haz": sp.stats.norm(1, 0.2),
    "f_haz": sp.stats.randint(low=0, high=len(f_haz_list)),
}
haz_iv = InputVar(haz_func, haz_distr)


def impf_func(G=1, v_half=84.7, vmin=25.7, k=3, _id=1):

    def xhi(v, v_half, vmin):
        return max([(v - vmin), 0]) / (v_half - vmin)

    def sigmoid_func(v, G, v_half, vmin, k):
        return G * xhi(v, v_half, vmin) ** k / (1 + xhi(v, v_half, vmin) ** k)

    # In-function imports needed only for parallel computing on Windows
    import numpy as np
    from climada.entity import ImpactFunc, ImpactFuncSet
```

(continues on next page)

(continued from previous page)

```
    imp_fun = ImpactFunc()
    imp_fun.haz_type = "WS"
    imp_fun.id = _id
    imp_fun.intensity_unit = "m/s"
    imp_fun.intensity = np.linspace(0, 150, num=100)
    imp_fun.mdd = np.repeat(1, len(imp_fun.intensity))
    imp_fun.paa = np.array(
        [sigmoid_func(v, G, v_half, vmin, k) for v in imp_fun.intensity]
    )
    imp_fun.check()
    impf_set = ImpactFuncSet()
    impf_set.append(imp_fun)
    return impf_set


impf_distr = {
    "G": sp.stats.truncnorm(0.5, 1.5),
    "v_half": sp.stats.uniform(35, 65),
    "vmin": sp.stats.uniform(0, 15),
    "k": sp.stats.uniform(1, 4),
}
impf_iv = InputVar(impf_func, impf_distr)
```

```
from climada.engine.unsequa import CalcImpact

calc_imp = CalcImpact(exp_iv, impf_iv, haz_iv)
```

Now that the samples have been generated, it is crucial to oder the samples in order to minimize the number of times files have to be loaded. In this case, loading the hazards take more time than loading the exposures. We thus sort first by hazards (which then each have to be loaded one single time), and then by exposures (which have to be each loaded once for each hazard).

```
# Ordering of the samples by hazard first and exposures second
output_imp = calc_imp.make_sample(N=2**2, sampling_kwargs={"skip_values": 2**3})
output_imp.order_samples(by=["f_haz", "f_exp"])
```

We can verify how the samples are ordered. In the graph below, it is confirmed that the hazard are ordered, and thus the hazards will be loaded once each. The exposures on the other changes at most once per hazard.

```
import matplotlib.pyplot as plt

e = output_imp.samples_df["f_exp"].values
h = output_imp.samples_df["f_haz"].values
```

Note that due to the very small number of samples chosen here for illustrative purposes, not all combinations of hazard and exposures are part of the samples. This is due to the nature of the Sobol sequence (default sampling method).

```
plt.plot(e, label="exposures")
plt.plot(h, label="hazards")
plt.xlabel("samples")
plt.ylabel("file number")
```

(continues on next page)

(continued from previous page)

```
plt.title("Order of exposures and hazards files in samples")
plt.legend(loc="upper right");
```

```
output_imp = calc_imp.uncertainty(output_imp)
```

2.7.2 Helper methods for InputVar

This tutorial complements the general tutorial on the uncertainty and sensitivity analysis module *unsequa*.

The InputVar class provides a few helper methods to generate generic uncertainty input variables for exposures, impact function sets, hazards, and entities (including measures cost and disc rates).

```
import warnings

warnings.filterwarnings("ignore")  # Ignore warnings for making the tutorial's pdf.
```

Exposures

The following types of uncertainties can be added:

- ET: scale the total value (homogeneously)

  The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_totvalue

- EN: mutliplicative noise (inhomogeneous)

  The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

- EL: sample uniformly from exposure list

  From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

If a bounds is None, this parameter is assumed to have no uncertainty.

Example: single exposures

```
# Define the base exposure
from climada.util.constants import EXP_DEMO_H5
from climada.entity import Exposures

exp_base = Exposures.from_hdf5(EXP_DEMO_H5)
```

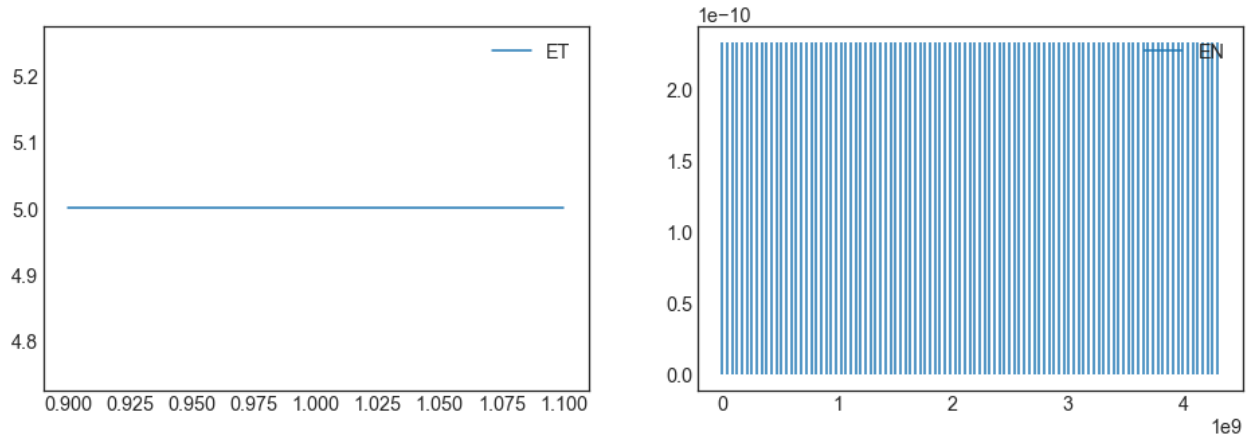
```
2022-07-07 15:13:32,000 - climada.entity.exposures.base - INFO - Reading /Users/
↪ckropf/climada/demo/data/exp_demo_today.h5
```

```
from climada.engine.unsequa import InputVar

bounds_totval = [0.9, 1.1]  # +- 10% noise on the total exposures value
bounds_noise = [0.9, 1.2]  # -10% - +20% noise each exposures point
exp_iv = InputVar.exp([exp_base], bounds_totval, bounds_noise)
```

```
# The difference in total value between the base exposure and the average input␣
↪uncertainty exposure
# due to the random noise on each exposures point (the average change in the total␣
↪value is 1.0).
avg_exp = exp_iv.evaluate()
(sum(avg_exp.gdf["value"]) - sum(exp_base.gdf["value"])) / sum(exp_base.gdf["value"])
```

```
0.03700231587024304
```

```
# The values for EN are seeds for the random number generator for the noise sampling␣
↪and
# thus are uniformly sampled numbers between (0, 2**32-1)
exp_iv.plot();
```



Example: list of litpop exposures with different exponents

```
# Define a generic method to make litpop instances with different exponent pairs.
from climada.entity import LitPop


def generate_litpop_base(
    impf_id, value_unit, haz, assign_centr_kwargs, choice_mn, **litpop_kwargs
):
    # In-function imports needed only for parallel computing on Windows
    from climada.entity import LitPop

    litpop_base = []
    for [m, n] in choice_mn:
        print("\n Computing litpop for m=%d, n=%d \n" % (m, n))
        litpop_kwargs["exponents"] = (m, n)
        exp = LitPop.from_countries(**litpop_kwargs)
        exp.gdf["impf_" + haz.haz_type] = impf_id
        exp.gdf.drop("impf_", axis=1, inplace=True)
        if value_unit is not None:
            exp.value_unit = value_unit
        exp.assign_centroids(haz, **assign_centr_kwargs)
        litpop_base.append(exp)
```

(continues on next page)

(continued from previous page)

```
    return litpop_base
```

```
# Define the parameters of the LitPop instances
tot_pop = 11.317e6
impf_id = 1
value_unit = "people"
litpop_kwargs = {
    "countries": ["CUB"],
    "res_arcsec": 150,
    "reference_year": 2020,
    "fin_mode": "norm",
    "total_values": [tot_pop],
}
assign_centr_kwargs = {}

# The hazard is needed to assign centroids
from climada.util.constants import HAZ_DEMO_H5
from climada.hazard import Hazard

haz = Hazard.from_hdf5(HAZ_DEMO_H5)
```

```
2022-07-07 15:13:32,787 - climada.hazard.base - INFO - Reading /Users/ckropf/climada/
↪demo/data/tc_fl_1990_2004.h5
```

```
# Generate the LitPop list

choice_mn = [[0, 0.5], [0, 1], [0, 2]]  # Choice of exponents m,n

litpop_list = generate_litpop_base(
    impf_id, value_unit, haz, assign_centr_kwargs, choice_mn, **litpop_kwargs
)
```

```
 Computing litpop for m=0, n=0

2022-07-07 15:13:33,055 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:13:34,051 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,082 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,109 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,135 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,163 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,188 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,223 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:13:34,289 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,316 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,325 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:34,326 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,355 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,385 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,410 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,435 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,459 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,487 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,508 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,519 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:34,520 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,543 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,570 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,597 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,621 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,645 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,685 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,710 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,742 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,768 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,791 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,830 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,862 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,899 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:13:34,933 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,955 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:34,981 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,019 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,043 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,068 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,090 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,119 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,142 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:35,173 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:13:35,173 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:13:35,174 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:13:35,174 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:13:35,175 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:13:35,179 - climada.entity.exposures.base - INFO - Matching 5524␣
↪exposures with 2500 centroids.
2022-07-07 15:13:35,181 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:13:35,189 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 332 coordinates.

 Computing litpop for m=0, n=1

2022-07-07 15:13:35,416 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:13:36,256 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,284 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,309 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,335 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,367 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,392 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,424 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,494 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:13:36,519 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,529 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:36,530 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,556 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,586 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,610 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,637 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,666 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,691 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,716 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,725 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:36,726 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,754 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,786 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,818 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,843 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,872 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,909 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,936 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:36,965 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

```
2022-07-07 15:13:36,989 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,013 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,052 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,087 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,123 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,156 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:13:37,177 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,201 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,241 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,263 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,288 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,311 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,343 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,367 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:37,400 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:13:37,400 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:13:37,401 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:13:37,401 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:13:37,402 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:13:37,406 - climada.entity.exposures.base - INFO - Matching 5524␣
↪exposures with 2500 centroids.
2022-07-07 15:13:37,407 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:13:37,415 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 332 coordinates.

 Computing litpop for m=0, n=2

2022-07-07 15:13:37,637 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:13:38,561 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,589 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,615 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,638 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,665 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,689 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,720 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,784 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,808 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:13:38,820 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:38,820 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,844 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,873 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,897 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,920 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,944 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,968 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,990 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:38,999 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:13:39,000 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,022 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,047 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,074 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,097 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,123 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,161 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,187 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,217 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,242 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,265 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,304 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,334 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,370 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,402 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,423 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:13:39,448 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,487 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,510 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,532 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,554 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,580 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,603 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:13:39,633 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:13:39,634 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:13:39,634 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:13:39,635 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:13:39,635 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:13:39,639 - climada.entity.exposures.base - INFO - Matching 5524␣
↪exposures with 2500 centroids.
2022-07-07 15:13:39,641 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:13:39,649 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 332 coordinates.
```

```
from climada.engine.unsequa import InputVar

bounds_totval = [0.9, 1.1]  # +- 10% noise on the total exposures value
litpop_iv = InputVar.exp(exp_list=litpop_list, bounds_totval=bounds_totval)
```

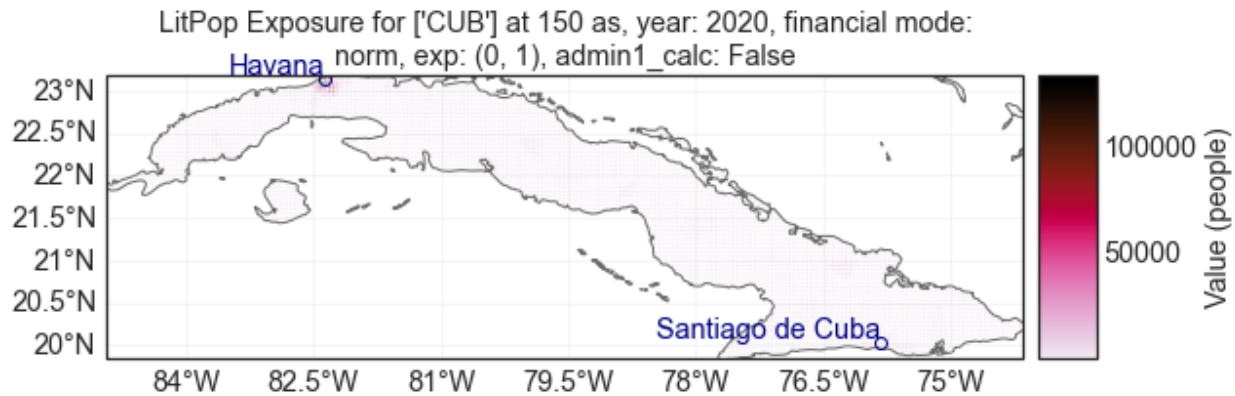
```
# To choose n=0.5, we have to set EL=1 (the index of 0.5 in choice_n = [0, 0.5, 1, 2])
pop_half = litpop_iv.evaluate(ET=1, EL=1)
```

```
pop_half.gdf.tail()
```

```
           value                    geometry   latitude  longitude  region_id  \
5519   92.974926  POINT (-80.52083 23.18750)  23.187500 -80.520833        192
5520  131.480741  POINT (-80.47917 23.18750)  23.187500 -80.479167        192
5521   77.695093  POINT (-80.68750 23.18750)  23.187500 -80.687500        192
5522   43.122163  POINT (-80.89583 23.14583)  23.145833 -80.895833        192
5523  106.033524  POINT (-80.85417 23.14583)  23.145833 -80.854167        192

      impf_TC  centr_TC
5519        1       619
5520        1       619
5521        1       618
5522        1       617
5523        1       617
```

```
pop_half.plot_hexbin();
```

```
2022-07-07 15:13:39,690 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0],UNIT[
↪"degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS[
↪"Longitude",EAST],AUTHORITY["EPSG","4326"]]'. Using projection PlateCarree in plot.
```



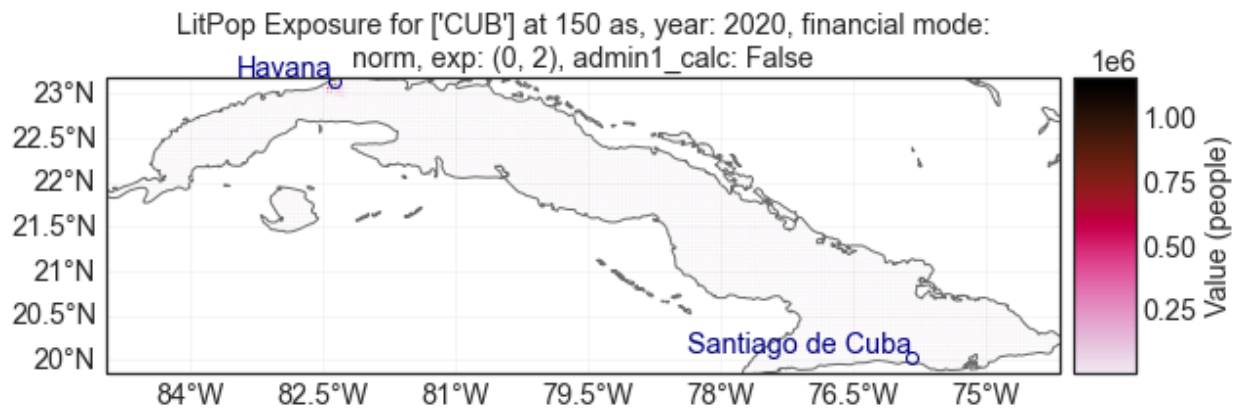

```
# To choose n=1, we have to set EL=2 (the index of 1 in choice_n = [0, 0.5, 1, 2])
pop_one = litpop_iv.evaluate(ET=1, EL=2)
```

```
pop_one.gdf.tail()
```

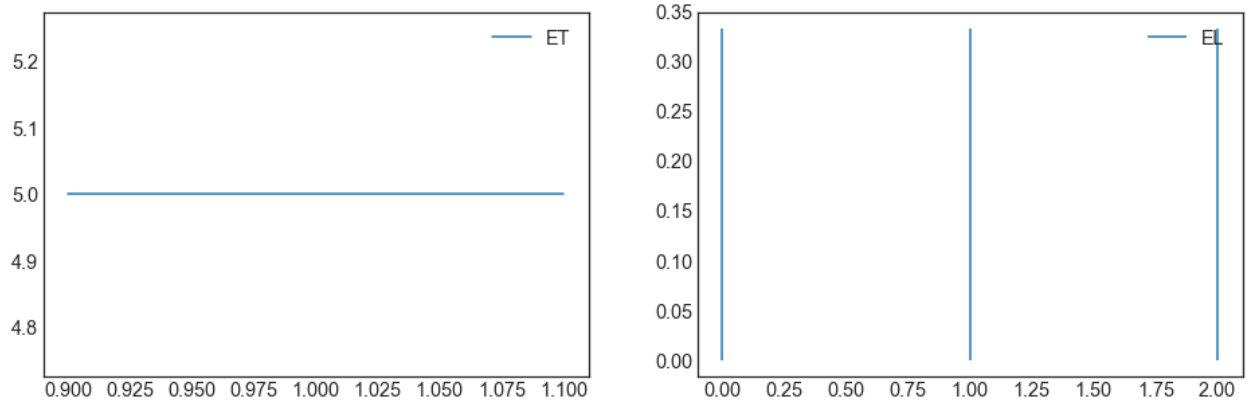
```
         value                     geometry   latitude  longitude  region_id  \
5519  0.567593  POINT (-80.52083 23.18750)  23.187500 -80.520833        192
5520  1.135089  POINT (-80.47917 23.18750)  23.187500 -80.479167        192
5521  0.396363  POINT (-80.68750 23.18750)  23.187500 -80.687500        192
5522  0.122097  POINT (-80.89583 23.14583)  23.145833 -80.895833        192
5523  0.738231  POINT (-80.85417 23.14583)  23.145833 -80.854167        192

      impf_TC  centr_TC
5519        1       619
5520        1       619
5521        1       618
5522        1       617
5523        1       617
```

```
pop_one.plot_hexbin();
```

```
2022-07-07 15:13:45,584 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0],UNIT[
↪"degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS[
↪"Longitude",EAST],AUTHORITY["EPSG","4326"]]'. Using projection PlateCarree in plot.
```




```
# The values for EN are seeds for the random number generator for the noise sampling
↪and
# thus are uniformly sampled numbers between (0, 2**32-1)
litpop_iv.plot();
```



Hazard

The following types of uncertainties can be added:

- HE: sub-sampling events from the total event set

  For each sub-sample, n_ev events are sampled with replacement. HE is the value of the seed for the uniform random number generator.

- HI: scale the intensity of all events (homogeneously)

  The instensity of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_int

- HA: scale the fraction of all events (homogeneously)

  The fraction of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_frac

- HF: scale the frequency of all events (homogeneously)

  The frequency of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_freq

- HL: sample uniformly from hazard list

From the provided list of hazard is elements are uniformly sampled. For example, Hazards outputs from dynamical models for different input factors.

If a bounds is None, this parameter is assumed to have no uncertainty.

```
# Define the base exposure
from climada.util.constants import HAZ_DEMO_H5
from climada.hazard import Hazard

haz_base = Hazard.from_hdf5(HAZ_DEMO_H5)
```

```
2022-07-07 15:13:51,145 - climada.hazard.base - INFO - Reading /Users/ckropf/climada/
↪demo/data/tc_fl_1990_2004.h5
```

```
from climada.engine.unsequa import InputVar

bounds_freq = [0.9, 1.1]  # +- 10% noise on the frequency of all events
bounds_int = None  # No uncertainty on the intensity
n_ev = None
haz_iv = InputVar.haz(
    [haz_base], n_ev=n_ev, bounds_freq=bounds_freq, bounds_int=bounds_int
)
```

```
# The difference in frequency for HF=1.1 is indeed 10%.
haz_high_freq = haz_iv.evaluate(HE=n_ev, HI=None, HF=1.1)
(sum(haz_high_freq.frequency) - sum(haz_base.frequency)) / sum(haz_base.frequency)
```

```
0.10000000000000736
```

```
bounds_freq = [0.9, 1.1]  # +- 10% noise on the frequency of all events
bounds_int = None  # No uncertainty on the intensity
bounds_frac = [0.7, 1.1]  # noise on the fraction of all events
n_ev = round(
    0.8 * haz_base.size
)  # sub-sample with re-draw events to obtain hazards with n=0.8*tot_number_events
haz_iv = InputVar.haz(
    [haz_base],
    n_ev=n_ev,
    bounds_freq=bounds_freq,
    bounds_int=bounds_int,
    bounds_frac=bounds_frac,
)
```

Note that the HE is not a univariate distribution, but for each sample corresponds to the names of the sub-sampled events. However, to simplify the data stream, the HE is saved as the seed for the random number generator that made the sample. Hence, the value of HE is a label for the given sample. If really needed, the exact chosen events can be obtained as follows.

```
import numpy as np

HE = 2618981871  # The random seed (number between 0 and 2**32)
rng = np.random.RandomState(int(HE))  # Initialize a random state with the seed
chosen_ev = list(
```

(continues on next page)

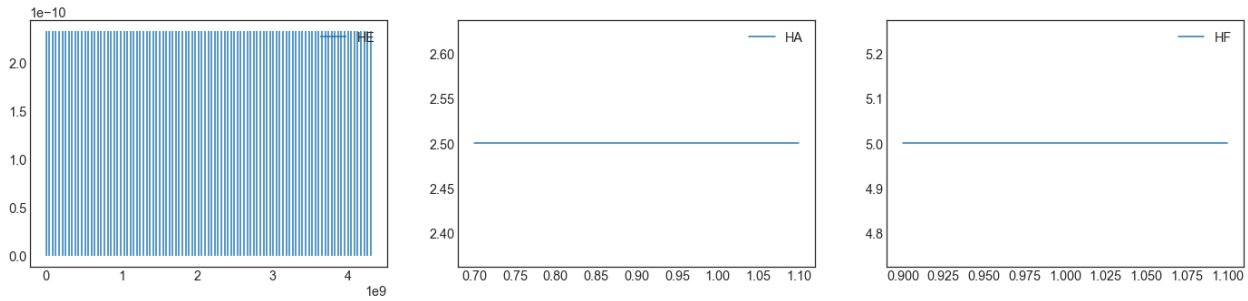
(continued from previous page)

```
    rng.choice(haz_base.event_name, int(n_ev))
)  # Obtain the corresponding events
```

```
# The first event is
chosen_ev[0]
```

```
'1998209N11335'
```

```
# The values for HE are seeds for the random number generator for the noise sampling␣
↪and
# thus are uniformly sampled numbers between (0, 2**32-1)
haz_iv.plot();
```



The number of events per sub-sample is equal to n_ev

```
# The number of events per sample is equal to n_ev
haz_sub = haz_iv.evaluate(HE=928165924, HI=None, HF=1.1, HA=None)
# The number for HE is irrelevant, as all samples have the same n_Ev
haz_sub.size - n_ev
```

```
0
```

ImpactFuncSet

The following types of uncertainties can be added:

- MDD: scale the mdd (homogeneously)

  The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

- PAA: scale the paa (homogeneously)

  The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

- IFi: shift the intensity (homogeneously)

  The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_int

- IL: sample uniformly from impact function set list

  From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

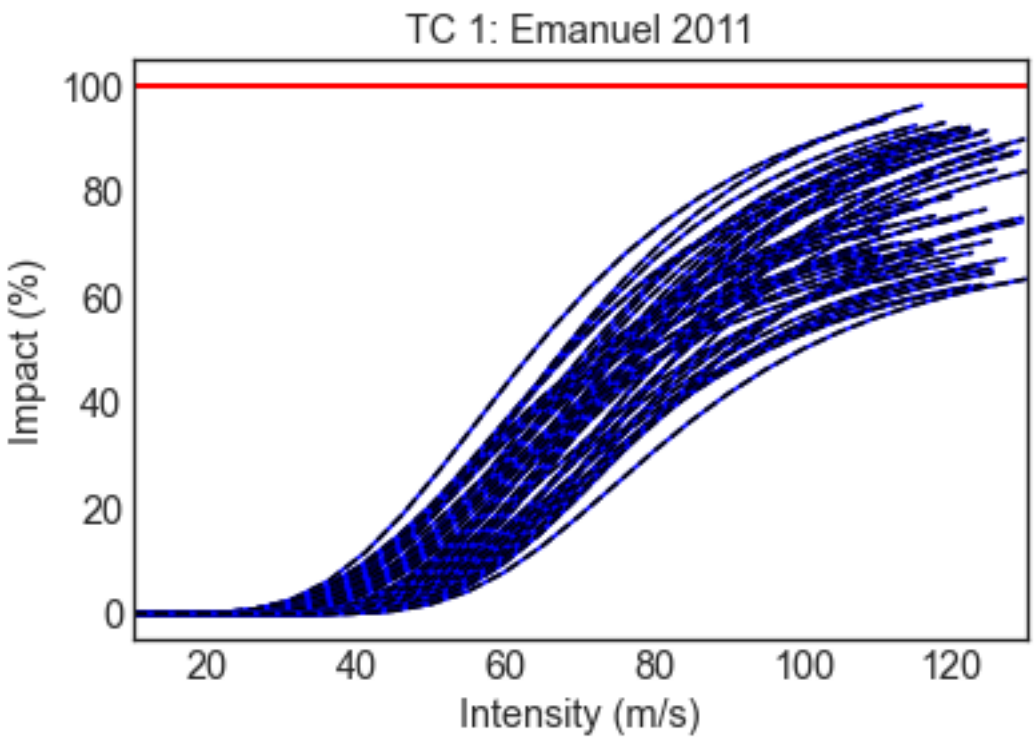
If a bounds is None, this parameter is assumed to have no uncertainty.

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone

impf = ImpfTropCyclone.from_emanuel_usa()
impf_set_base = ImpactFuncSet([impf])
```

It is necessary to specify the hazard type and the impact function id. For simplicity, the default uncertainty input variable only looks at the uncertainty on one single impact function.

```
from climada.engine.unsequa import InputVar

bounds_impfi = [-10, 10]  # -10 m/s ; +10m/s uncertainty on the intensity
bounds_mdd = [0.7, 1.1]  # -30% - +10% uncertainty on the mdd
bounds_paa = None  # No uncertainty in the paa
impf_iv = InputVar.impfset(
    impf_set_list=[impf_set_base],
    bounds_impfi=bounds_impfi,
    bounds_mdd=bounds_mdd,
    bounds_paa=bounds_paa,
    haz_id_dict={"TC": [1]},
)
```

```
# Plot the impact function for 50 random samples (note for the expert, these are not␣
↪global)
n = 50
ax = impf_iv.evaluate().plot()
inten = impf_iv.distr_dict["IFi"].rvs(size=n)
mdd = impf_iv.distr_dict["MDD"].rvs(size=n)
for i, m in zip(inten, mdd):
    impf_iv.evaluate(IFi=i, MDD=m).plot(axis=ax)
ax.get_legend().remove()
```



Entity

The following types of uncertainties can be added:

- DR: value of constant discount rate (homogeneously)

  The value of the discounts in each year is sampled uniformly from a distribution with (min, max) = bounds_disc

- CO: scale the cost (homogeneously)

  The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = bounds_cost

- ET: scale the total value (homogeneously)

  The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_totval

- EN: mutliplicative noise (inhomogeneous)

  The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

- EL: sample uniformly from exposure list

  From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

- MDD: scale the mdd (homogeneously)

  The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

- PAA: scale the paa (homogeneously)

The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

- IFi: shift the intensity (homogeneously)

  The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_int

If a bounds is None, this parameter is assumed to have no uncertainty.

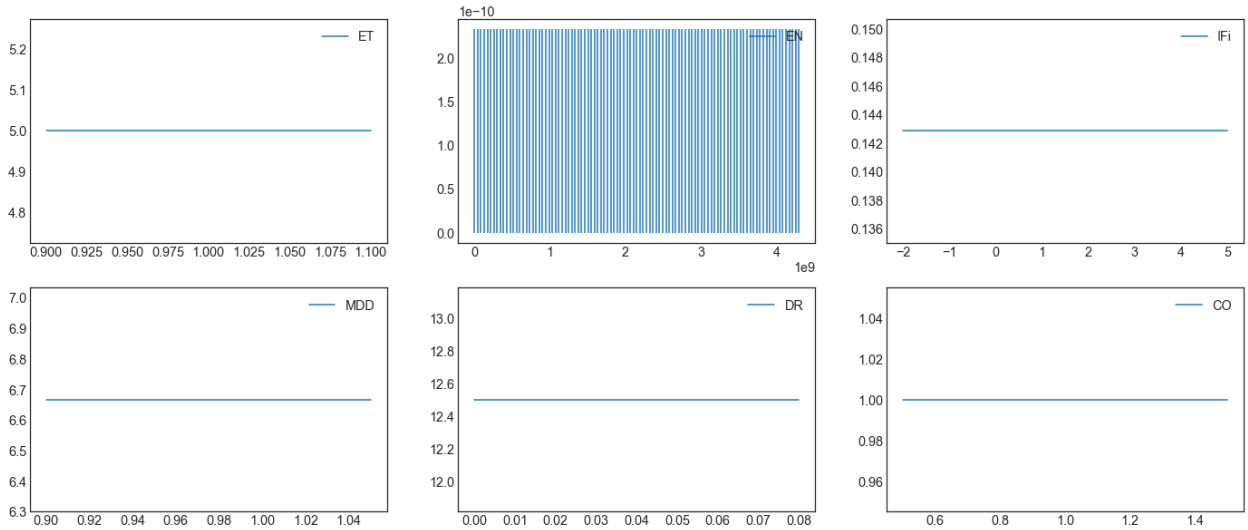
Example: single exposures

```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_TODAY

ent = Entity.from_excel(ENT_DEMO_TODAY)
ent.exposures.ref_year = 2018
ent.check()
```

```
2022-07-07 15:18:00,292 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:00,293 - climada.entity.exposures.base - INFO - geometry not set.
2022-07-07 15:18:00,294 - climada.entity.exposures.base - INFO - region_id not set.
2022-07-07 15:18:00,294 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
from climada.engine.unsequa import InputVar

ent_iv = InputVar.ent(
    impf_set_list=[ent.impact_funcs],
    disc_rate=ent.disc_rates,
    exp_list=[ent.exposures],
    meas_set=ent.measures,
    bounds_disc=[0, 0.08],
    bounds_cost=[0.5, 1.5],
    bounds_totval=[0.9, 1.1],
    bounds_noise=[0.3, 1.9],
    bounds_mdd=[0.9, 1.05],
    bounds_paa=None,
    bounds_impfi=[-2, 5],
    haz_id_dict={"TC": [1]},
)
```

```
ent_iv.plot();
```



Example: list of Litpop exposures with different exponents

```
# Define a generic method to make litpop instances with different exponent pairs.
from climada.entity import LitPop


def generate_litpop_base(
    impf_id, value_unit, haz, assign_centr_kwargs, choice_mn, **litpop_kwargs
):
    # In-function imports needed only for parallel computing on Windows
    from climada.entity import LitPop

    litpop_base = []
    for [m, n] in choice_mn:
        print("\n Computing litpop for m=%d, n=%d \n" % (m, n))
        litpop_kwargs["exponents"] = (m, n)
        exp = LitPop.from_countries(**litpop_kwargs)
        exp.gdf["impf_" + haz.haz_type] = impf_id
        exp.gdf.drop("impf_", axis=1, inplace=True)
        if value_unit is not None:
            exp.value_unit = value_unit
        exp.assign_centroids(haz, **assign_centr_kwargs)
        litpop_base.append(exp)
    return litpop_base
```

```
# Define the parameters of the LitPop instances
impf_id = 1
value_unit = None
litpop_kwargs = {
    "countries": ["CUB"],
    "res_arcsec": 300,
    "reference_year": 2020,
}
assign_centr_kwargs = {}
```

(continues on next page)

(continued from previous page)

```
# The hazard is needed to assign centroids
from climada.util.constants import HAZ_DEMO_H5
from climada.hazard import Hazard

haz = Hazard.from_hdf5(HAZ_DEMO_H5)
```

```
2022-07-07 15:18:00,956 - climada.hazard.base - INFO - Reading /Users/ckropf/climada/
↪demo/data/tc_fl_1990_2004.h5
```

```
# Generate the LitPop list

choice_mn = [[1, 0.5], [0.5, 1], [1, 1]]  # Choice of exponents m,n

litpop_list = generate_litpop_base(
    impf_id, value_unit, haz, assign_centr_kwargs, choice_mn, **litpop_kwargs
)
```

```
 Computing litpop for m=1, n=0

2022-07-07 15:18:01,386 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:01,968 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:01,997 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,022 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,044 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,069 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,093 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,125 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,187 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,210 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,220 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,221 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,233 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,234 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,248 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,249 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:02,264 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,265 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,278 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,279 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,303 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,330 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,341 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,342 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,351 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,351 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,363 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,364 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,391 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,417 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,443 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,466 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,504 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,528 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,559 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,585 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,596 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,597 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,634 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,666 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,700 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,734 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:02,758 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,768 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,769 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,809 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,821 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,822 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,845 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,869 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,896 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,919 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:02,930 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:02,959 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:04,013 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:04,017 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:04,028 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:04,029 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:04,030 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:04,031 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:04,032 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:04,037 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:04,039 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:04,046 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.

 Computing litpop for m=0, n=1

2022-07-07 15:18:04,291 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:04,812 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,842 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,869 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:04,894 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,923 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,953 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:04,987 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,049 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,075 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,086 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,087 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,098 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
```

```
2022-07-07 15:18:05,099 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,114 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,115 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,131 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,132 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,148 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,149 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,174 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,199 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,208 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,209 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,220 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,221 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,233 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,234 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,261 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,287 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:05,310 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,333 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,368 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,392 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,420 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,444 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,455 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,456 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,492 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,523 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,557 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,588 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,611 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,622 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,623 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,659 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,671 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,671 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,693 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,716 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,742 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,764 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:05,775 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:05,801 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:06,617 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:06,621 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:06,630 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:06,631 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:06,631 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:06,632 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:06,632 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:06,636 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:06,637 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:06,643 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.

 Computing litpop for m=1, n=1

2022-07-07 15:18:06,884 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:07,423 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,449 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,473 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,496 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,521 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,544 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,574 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,637 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,660 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,670 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,670 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,682 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,683 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,697 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,698 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,710 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,711 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:07,723 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,724 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,748 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,773 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,783 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,784 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,793 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,794 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

```
2022-07-07 15:18:07,805 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:07,806 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,832 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,859 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,882 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,907 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,943 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,968 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:07,997 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,021 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,032 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:08,032 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,071 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,102 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,136 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,167 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,189 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,200 - climada.entity.exposures.litpop.litpop - INFO - No data␣
```

(continues on next page)

(continued from previous page)

```
↪point on destination grid within polygon.
2022-07-07 15:18:08,201 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,240 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,252 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:08,253 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,276 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,298 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,323 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,345 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:08,357 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:08,383 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:09,253 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:09,257 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:09,267 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:09,268 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:09,268 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:09,269 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:09,270 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:09,275 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:09,277 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:09,283 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.
```

```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_TODAY

ent = Entity.from_excel(ENT_DEMO_TODAY)
ent.exposures.ref_year = 2020
ent.check()
```

```
2022-07-07 15:18:09,400 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:09,401 - climada.entity.exposures.base - INFO - geometry not set.
2022-07-07 15:18:09,402 - climada.entity.exposures.base - INFO - region_id not set.
2022-07-07 15:18:09,402 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
from climada.engine.unsequa import InputVar
```

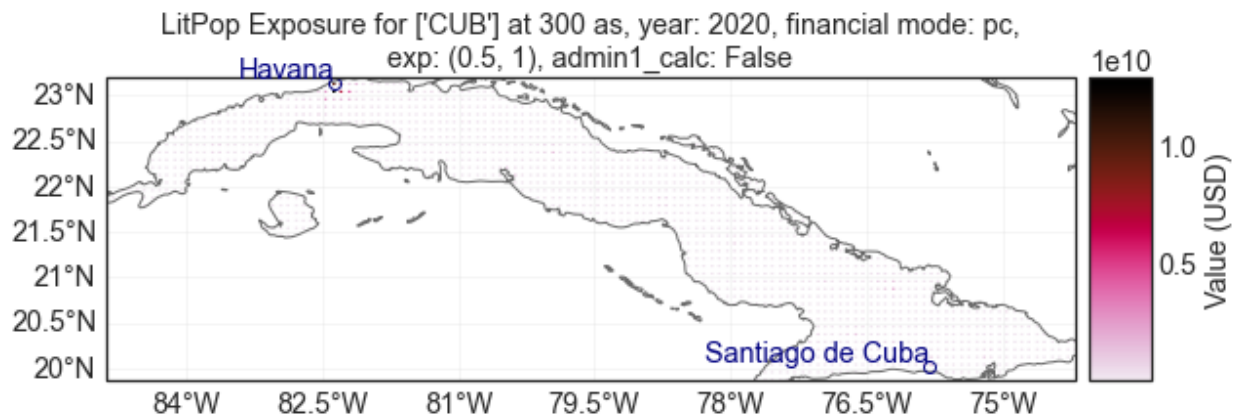
(continues on next page)

(continued from previous page)

```
ent_iv = InputVar.ent(
    impf_set_list=[ent.impact_funcs],
    disc_rate=ent.disc_rates,
    exp_list=litpop_list,
    meas_set=ent.measures,
    bounds_disc=[0, 0.08],
    bounds_cost=[0.5, 1.5],
    bounds_totval=[0.9, 1.1],
    bounds_noise=[0.3, 1.9],
    bounds_mdd=[0.9, 1.05],
    bounds_paa=None,
    bounds_impfi=[-2, 5],
    haz_id_dict={"TC": [1]},
)
```

```
ent_iv.evaluate().exposures.plot_hexbin();
```

```
2022-07-07 15:18:09,448 - climada.util.plot - WARNING - Error parsing coordinate␣
↪system 'GEOGCS["WGS 84",DATUM["WGS_1984",SPHEROID["WGS 84",6378137,298.257223563,
↪AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],PRIMEM["Greenwich",0],UNIT[
↪"degree",0.0174532925199433,AUTHORITY["EPSG","9122"]],AXIS["Latitude",NORTH],AXIS[
↪"Longitude",EAST],AUTHORITY["EPSG","4326"]]'. Using projection PlateCarree in plot.
```



Entity Future

The following types of uncertainties can be added:

- CO: scale the cost (homogeneously)

  The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = bounds_cost

- EG: scale the exposures growth (homogeneously)

  The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_eg

- EN: mutliplicative noise (inhomogeneous)

  The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number

generator.

- EL: sample uniformly from exposure list

  From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

- MDD: scale the mdd (homogeneously)

  The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

- PAA: scale the paa (homogeneously)

  The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

- IFi: shift the impact function intensity (homogeneously)

  The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_impfi

- IL: sample uniformly from impact function set list

  From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

If a bounds is None, this parameter is assumed to have no uncertainty.

Example: single exposures

```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_FUTURE

ent_fut = Entity.from_excel(ENT_DEMO_FUTURE)
ent_fut.exposures.ref_year = 2040
ent_fut.check()
```

```
2022-07-07 15:18:11,936 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:11,937 - climada.entity.exposures.base - INFO - geometry not set.
2022-07-07 15:18:11,938 - climada.entity.exposures.base - INFO - region_id not set.
2022-07-07 15:18:11,938 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
entfut_iv = InputVar.entfut(
    impf_set_list=[ent_fut.impact_funcs],
    exp_list=[ent_fut.exposures],
    meas_set=ent_fut.measures,
    bounds_cost=[0.6, 1.2],
    bounds_eg=[0.8, 1.5],
    bounds_noise=None,
    bounds_mdd=[0.7, 0.9],
    bounds_paa=[1.3, 2],
    haz_id_dict={"TC": [1]},
)
```

Example: list of exposures

```
# Define a generic method to make litpop instances with different exponent pairs.
from climada.entity import LitPop


def generate_litpop_base(
    impf_id, value_unit, haz, assign_centr_kwargs, choice_mn, **litpop_kwargs
):
    # In-function imports needed only for parallel computing on Windows
    from climada.entity import LitPop

    litpop_base = []
    for [m, n] in choice_mn:
        print("\n Computing litpop for m=%d, n=%d \n" % (m, n))
        litpop_kwargs["exponents"] = (m, n)
        exp = LitPop.from_countries(**litpop_kwargs)
        exp.gdf["impf_" + haz.haz_type] = impf_id
        exp.gdf.drop("impf_", axis=1, inplace=True)
        if value_unit is not None:
            exp.value_unit = value_unit
        exp.assign_centroids(haz, **assign_centr_kwargs)
        litpop_base.append(exp)
    return litpop_base
```

```
# Define the parameters of the LitPop instances
impf_id = 1
value_unit = None
litpop_kwargs = {
    "countries": ["CUB"],
    "res_arcsec": 300,
    "reference_year": 2040,
}
assign_centr_kwargs = {}

# The hazard is needed to assign centroids
from climada.util.constants import HAZ_DEMO_H5
from climada.hazard import Hazard

haz = Hazard.from_hdf5(HAZ_DEMO_H5)
```

```
2022-07-07 15:18:11,958 - climada.hazard.base - INFO - Reading /Users/ckropf/climada/
↪demo/data/tc_fl_1990_2004.h5
```

```
# Generate the LitPop list

choice_mn = [[1, 0.5], [0.5, 1], [1, 1]]  # Choice of exponents m,n

litpop_list = generate_litpop_base(
    impf_id, value_unit, haz, assign_centr_kwargs, choice_mn, **litpop_kwargs
)
```

```
 Computing litpop for m=1, n=0

2022-07-07 15:18:12,244 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:12,773 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,773 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,803 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,804 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,830 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,831 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,854 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,854 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,881 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,881 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,906 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,907 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:12,937 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:12,938 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,001 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,002 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,025 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,025 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,034 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,035 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,035 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,047 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,048 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,048 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:13,062 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,063 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,064 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,077 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,078 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,078 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,090 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,090 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,091 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,117 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,118 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,144 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,145 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,155 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,156 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,157 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,165 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,166 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,166 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,178 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,178 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,179 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,202 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,203 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,231 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,232 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:13,256 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,257 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,281 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,281 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,316 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,316 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,344 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,344 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,374 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,375 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,399 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,400 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,412 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,412 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
```

```
2022-07-07 15:18:13,413 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,451 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,452 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,483 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,483 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,521 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,522 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,554 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,555 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,579 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,579 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,590 - climada.entity.exposures.litpop.litpop - INFO - No data␣
```

(continues on next page)

(continued from previous page)

```
↪point on destination grid within polygon.
2022-07-07 15:18:13,591 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,592 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,630 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,630 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,643 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,644 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,644 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,666 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,667 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,690 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,691 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,718 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,718 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,742 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:13,742 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:13,754 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:13,780 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:14,384 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:14,388 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:14,396 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:14,397 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:14,397 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:14,398 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:14,398 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:14,401 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:14,403 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:14,409 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.

 Computing litpop for m=0, n=1
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:14,640 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:15,172 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,173 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,199 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,200 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,223 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,224 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,249 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,250 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,279 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,279 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,301 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,302 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,330 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,331 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,390 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,391 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,417 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,418 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,428 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,429 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,429 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,441 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,441 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,442 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:15,457 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,457 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,458 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

```
2022-07-07 15:18:15,471 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,472 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,472 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,483 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,484 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,485 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,512 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,513 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,539 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,540 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,550 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,551 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,552 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,562 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,562 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,563 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,575 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,575 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,576 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,601 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,602 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,633 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,634 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:15,661 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,662 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,688 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,689 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,729 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,730 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,757 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,758 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,787 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,787 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,812 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,813 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,824 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:15,824 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,825 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,862 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,862 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,893 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,894 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,928 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,929 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,961 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,962 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,984 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:15,985 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:15,999 - climada.entity.exposures.litpop.litpop - INFO - No data␣
```

(continues on next page)

(continued from previous page)

```
↪point on destination grid within polygon.
2022-07-07 15:18:16,000 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,000 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,040 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,040 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,051 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:16,052 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,052 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,076 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,077 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,103 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,103 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,131 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,132 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,156 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:16,156 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:16,167 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
```

```
2022-07-07 15:18:16,195 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:17,186 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:17,190 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:17,200 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:17,201 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:17,201 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:17,202 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:17,203 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:17,208 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:17,210 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:17,216 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.
```

(continues on next page)

(continued from previous page)

```
 Computing litpop for m=1, n=1

2022-07-07 15:18:17,454 - climada.entity.exposures.litpop.litpop - INFO -
 LitPop: Init Exposure for country: CUB (192)...

2022-07-07 15:18:17,967 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:17,967 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:17,997 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:17,998 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,024 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,024 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,050 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,050 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,079 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,080 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,111 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,111 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,155 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,155 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,223 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,224 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,250 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,251 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,261 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,262 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,263 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,275 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,275 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,276 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:18,291 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,291 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,292 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,304 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,305 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,305 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,316 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,317 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,317 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,344 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,345 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,371 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,372 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,385 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,386 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,387 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,399 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,399 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,400 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,413 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,414 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,414 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,440 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,441 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,468 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,469 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
```

(continues on next page)

(continued from previous page)

```
↪Version v4.11
2022-07-07 15:18:18,492 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,493 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,516 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,516 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,553 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
```

```
2022-07-07 15:18:18,553 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,577 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,578 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,607 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,607 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,631 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,632 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,644 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,644 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,645 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,681 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,682 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,714 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,714 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,748 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,748 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,782 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,782 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,804 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,804 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
```

(continues on next page)

(continued from previous page)

```
2022-07-07 15:18:18,816 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,817 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,818 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,855 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,856 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,866 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:18,867 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,867 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,891 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,891 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,914 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,914 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,942 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,943 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,965 - climada.entity.exposures.litpop.gpw_population - WARNING -␣
↪Reference year: 2040. Using nearest available year for GPW data: 2020
2022-07-07 15:18:18,966 - climada.entity.exposures.litpop.gpw_population - INFO - GPW␣
↪Version v4.11
2022-07-07 15:18:18,978 - climada.entity.exposures.litpop.litpop - INFO - No data␣
↪point on destination grid within polygon.
2022-07-07 15:18:19,006 - climada.util.finance - WARNING - No data available for␣
↪country. Using non-financial wealth instead
2022-07-07 15:18:19,855 - climada.util.finance - INFO - GDP CUB 2020: 1.074e+11.
2022-07-07 15:18:19,859 - climada.util.finance - WARNING - No data for country, using␣
↪mean factor.
2022-07-07 15:18:19,869 - climada.entity.exposures.base - INFO - Hazard type not set␣
↪in impf_
2022-07-07 15:18:19,870 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:19,870 - climada.entity.exposures.base - INFO - cover not set.
2022-07-07 15:18:19,871 - climada.entity.exposures.base - INFO - deductible not set.
2022-07-07 15:18:19,872 - climada.entity.exposures.base - INFO - centr_ not set.
2022-07-07 15:18:19,875 - climada.entity.exposures.base - INFO - Matching 1388␣
↪exposures with 2500 centroids.
2022-07-07 15:18:19,878 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2022-07-07 15:18:19,884 - climada.util.coordinates - WARNING - Distance to closest␣
↪centroid is greater than 100km for 78 coordinates.
```

```
from climada.entity import Entity
from climada.util.constants import ENT_DEMO_FUTURE

ent_fut = Entity.from_excel(ENT_DEMO_FUTURE)
ent_fut.exposures.ref_year = 2040
ent_fut.check()
```

```
2022-07-07 15:18:19,989 - climada.entity.exposures.base - INFO - category_id not set.
2022-07-07 15:18:19,989 - climada.entity.exposures.base - INFO - geometry not set.
2022-07-07 15:18:19,990 - climada.entity.exposures.base - INFO - region_id not set.
2022-07-07 15:18:19,990 - climada.entity.exposures.base - INFO - centr_ not set.
```

```
from climada.engine.unsequa import InputVar

entfut_iv = InputVar.entfut(
    impf_set_list=[ent_fut.impact_funcs],
    exp_list=litpop_list,
    meas_set=ent_fut.measures,
    bounds_cost=[0.6, 1.2],
    bounds_eg=[0.8, 1.5],
    bounds_noise=None,
    bounds_mdd=[0.7, 0.9],
    bounds_paa=[1.3, 2],
    haz_id_dict={"TC": [1]},
)
```

2.8 Forecast class

This class deals with weather forecasts and uses CLIMADA ImpactCalc.impact() to forecast impacts of weather events on society. It mainly does one thing:

- it contains all plotting and other functionality that are specific for weather forecasts, impact forecasts and warnings

The class is different from the Impact class especially because features of the Impact class like Exceedence frequency curves, annual average impact etc, do not make sense if the hazard is e.g. a 5 day weather forecast. As the class is relatively new, there might be future changes to the datastructure, the methods, and the parameters used to call the methods.

2.8.1 Example: forecast of building damages due to wind in Switzerland

Before using the forecast class, hazard, exposure and vulnerability need to be created. The hazard looks at the weather forecast from today for an event with two days lead time (meaning the day after tomorrow). generate_WS_forecast_hazard is used to download a current weather forecast for wind gust from opendata.dwd.de. An Impact function for building damages due to storms is created. And with only a few lines of code, a LitPop exposure for Switzerland is generated, and the impact is calculated with a default impact function. With a further line of code, the mean damage per grid point for the day after tomorrow is plotted on a map.

```
from datetime import datetime
from cartopy import crs as ccrs

from climada.util.config import CONFIG
from climada.engine.forecast import Forecast
from climada.hazard.storm_europe import StormEurope, generate_WS_forecast_hazard
```

(continues on next page)

(continued from previous page)

```
from climada.entity.impact_funcs.storm_europe import ImpfStormEurope
from climada.entity import ImpactFuncSet
from climada.entity import LitPop
```

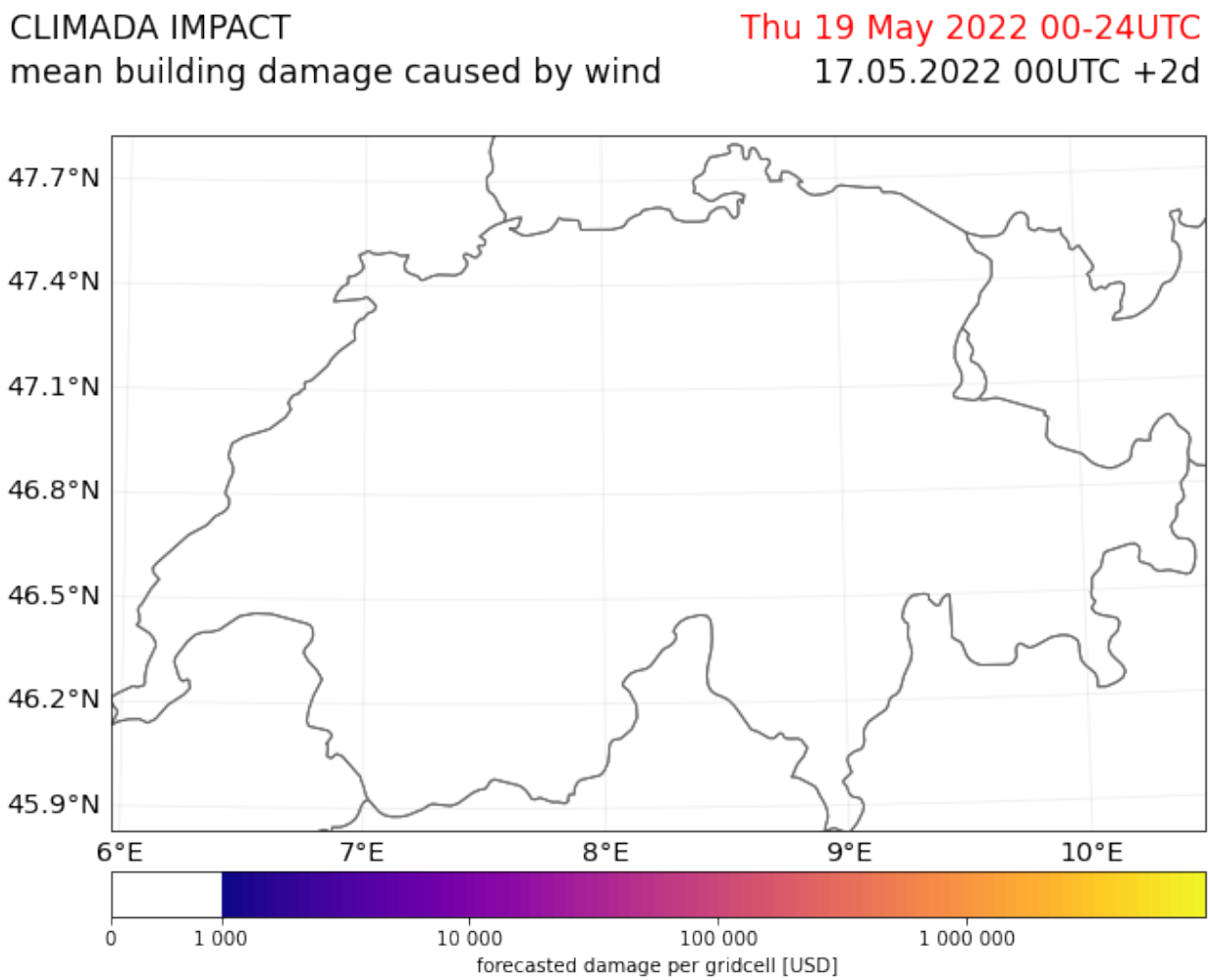
```
# generate hazard
hazard, haz_model, run_datetime, event_date = generate_WS_forecast_hazard()
# generate hazard with forecasts from past dates (works only if the files have␣
↪already been downloaded)
# hazard, haz_model, run_datetime, event_date = generate_WS_forecast_hazard(
#     run_datetime=datetime(2022,5,17),
#     event_date=datetime(2022,5,19))
```

```
# generate vulnerability
impact_function = ImpfStormEurope.from_welker()
impact_function_set = ImpactFuncSet([impact_function])
```

```
# generate exposure and save to file
filename_exp = CONFIG.local_data.save_dir.dir() / ("exp_litpop_Switzerland.hdf5")
if filename_exp.exists():
    exposure = LitPop.from_hdf5(filename_exp)
else:
    exposure = LitPop.from_countries("Switzerland", reference_year=2020)
    exposure.write_hdf5(filename_exp)
```

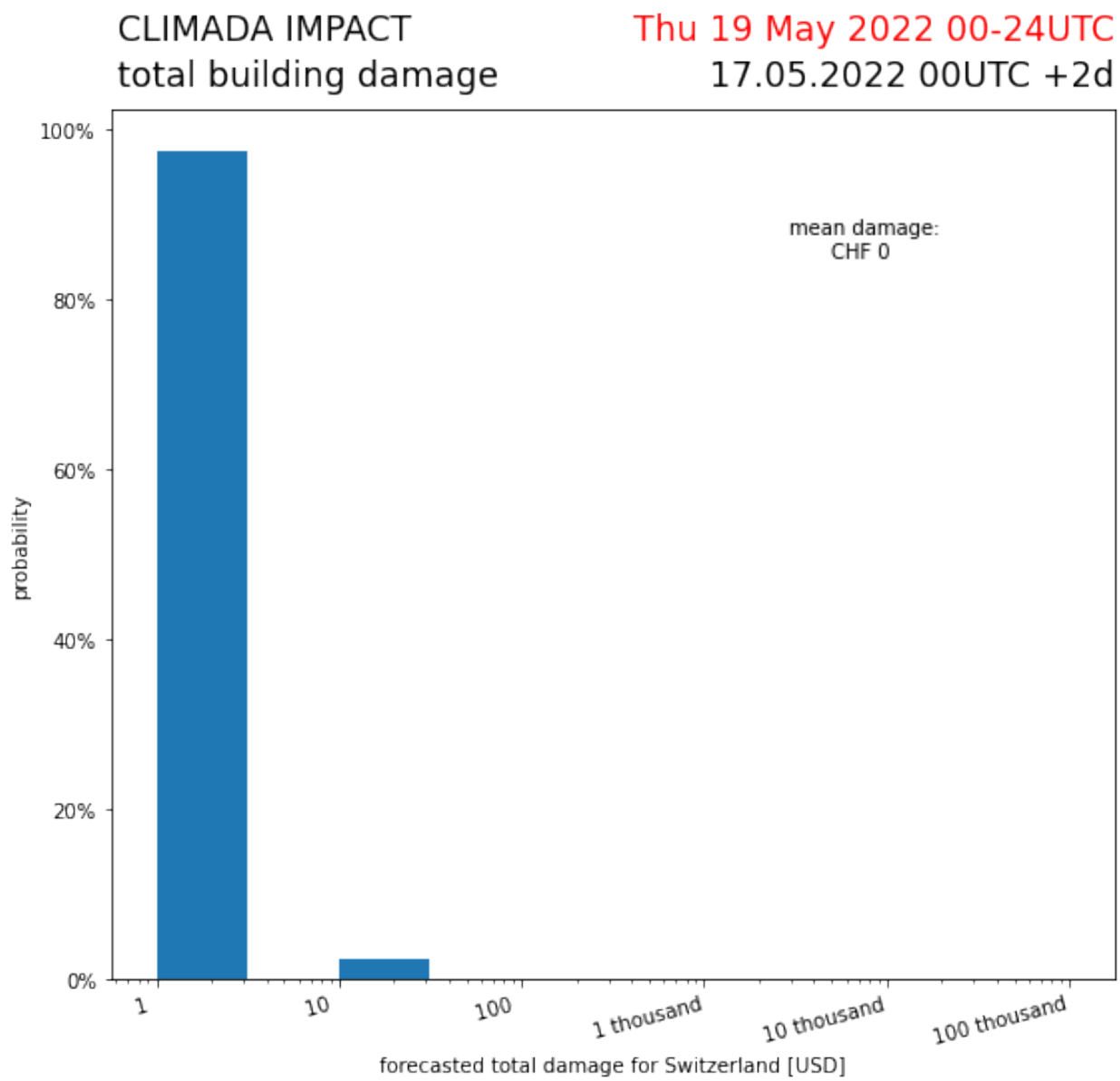
```
# create and calculate Forecast
CH_WS_forecast = Forecast({run_datetime: hazard}, exposure, impact_function_set)
CH_WS_forecast.calc()
```

```
CH_WS_forecast.plot_imp_map(save_fig=False, close_fig=False, proj=ccrs.epsg(2056));
```



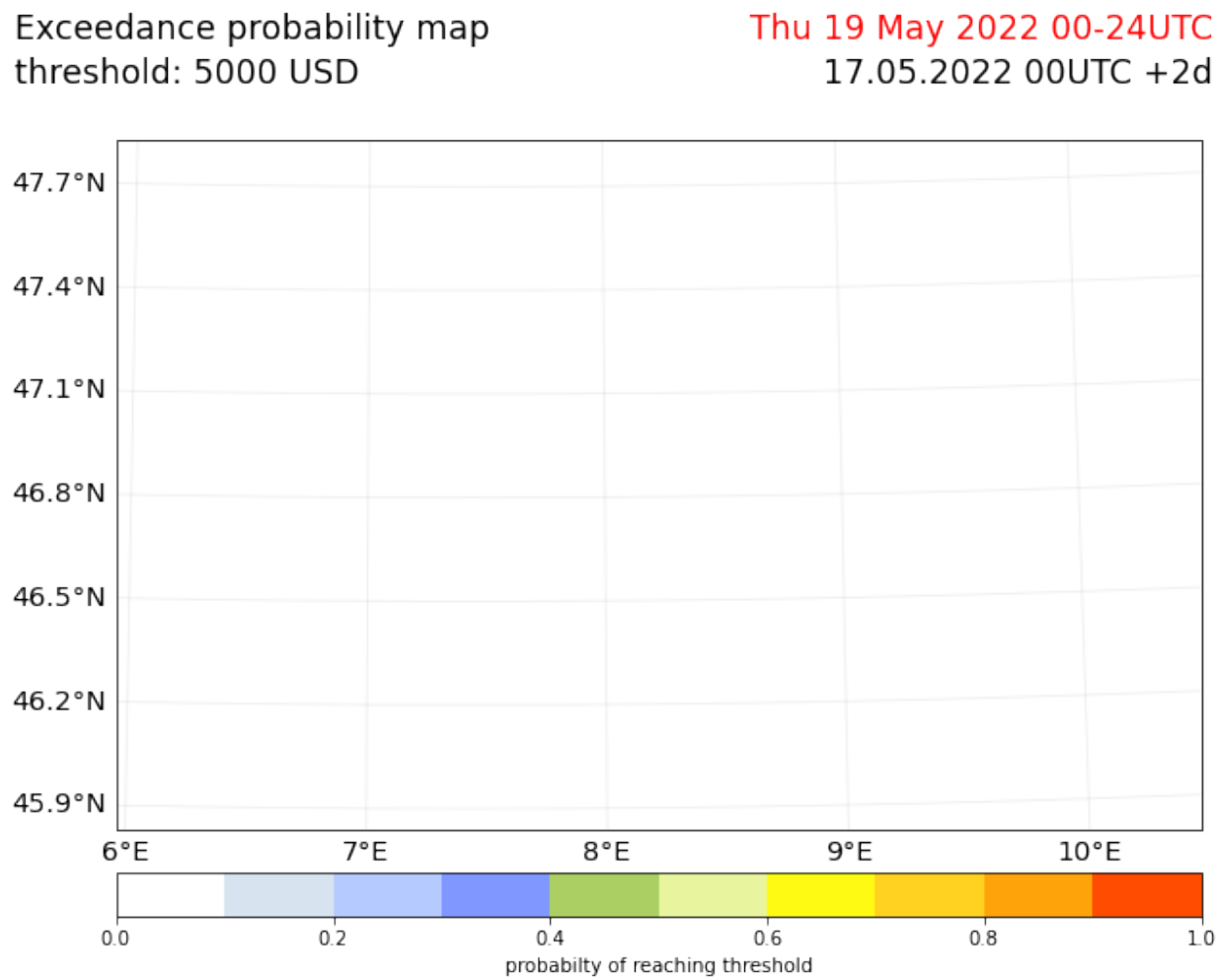
Here you see a different plot highlighting the spread of the impact forecast calculated from the different ensemble members of the weather forecast.

```
CH_WS_forecast.plot_hist(save_fig=False, close_fig=False);
```



It is possible to color the pixels depending on the probability that a certain threshold of impact is reach at a certain grid point

```
CH_WS_forecast.plot_exceedence_prob(
    threshold=5000, save_fig=False, close_fig=False, proj=ccrs.epsg(2056)
);
```



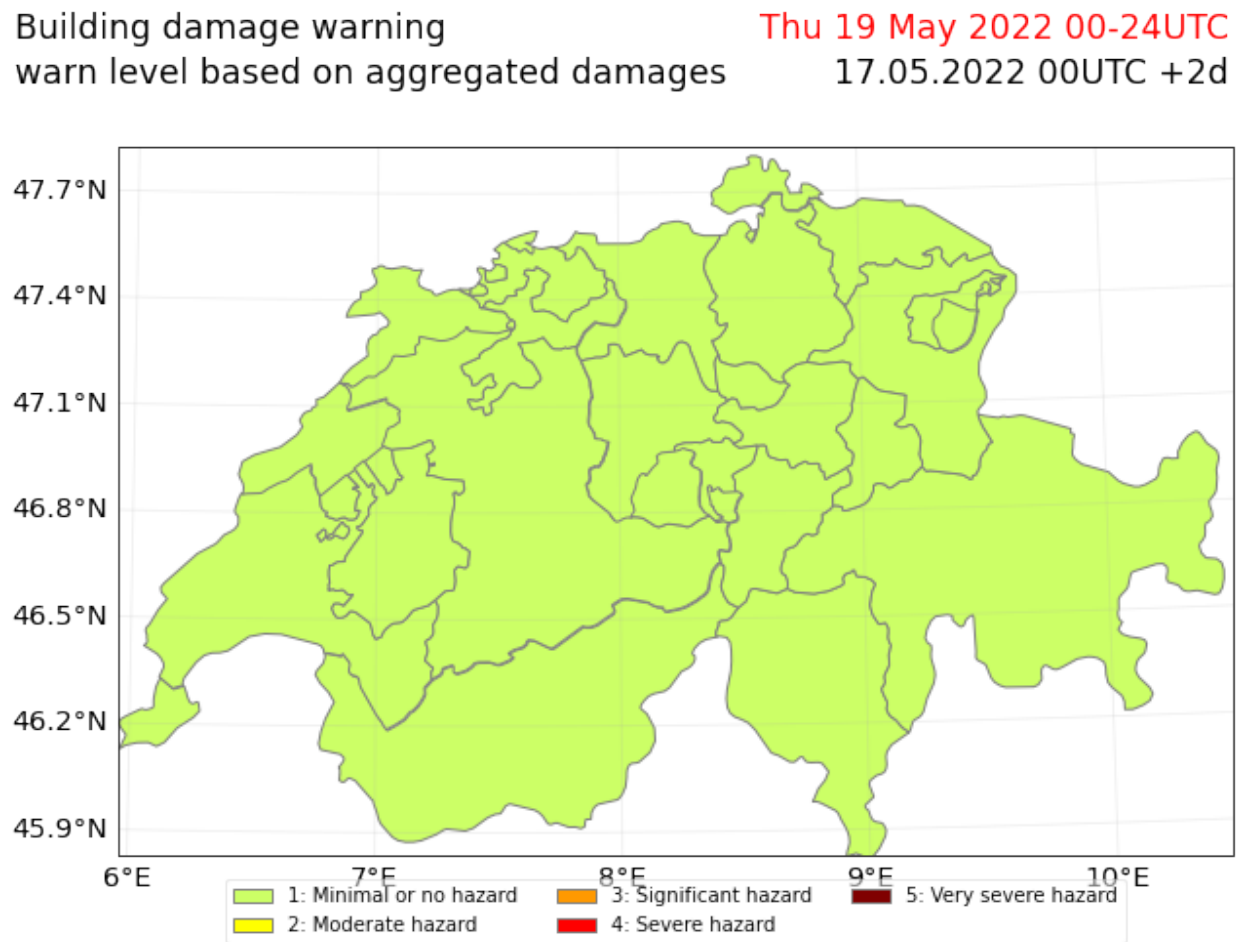
It is possible to color the cantons of Switzerland with warning colors, based on aggregated forecasted impacts in their area.

```
import fiona
from cartopy.io import shapereader
from climada.util.config import CONFIG


# create a file containing the polygons of Swiss cantons using natural earth
cantons_file = CONFIG.local_data.save_dir.dir() / "cantons.shp"
adm1_shape_file = shapereader.natural_earth(
    resolution="10m", category="cultural", name="admin_1_states_provinces"
)
if not cantons_file.exists():
    with fiona.open(adm1_shape_file, "r") as source:
        with fiona.open(cantons_file, "w", **source.meta) as sink:

            for f in source:
                if f["properties"]["adm0_a3"] == "CHE":
                    sink.write(f)
CH_WS_forecast.plot_warn_map(
```

(continues on next page)

(continued from previous page)

```
    str(cantons_file),
    decision_level="polygon",
    thresholds=[100000, 500000, 1000000, 5000000],
    probability_aggregation="mean",
    area_aggregation="sum",
    title="Building damage warning",
    explain_text="warn level based on aggregated damages",
    save_fig=False,
    close_fig=False,
    proj=ccrs.epsg(2056),
);
```



2.8.2 Example 2: forecast of wind warnings in Switzerland

Instead of a fully fledged socio-economic impact of storms, one can also simplify the hazard, exposure, vulnerability model, by looking at a "neutral" exposure (=1 at every gridpoint) and using a step function as impact function to arrive at warn levels. It also shows how the attributes hazard, exposure or vulnerability can be set before calling calc(), and are then considered in the forecast instead of the defined defaults.

```
from pandas import DataFrame
import numpy as np
```

(continues on next page)

(continued from previous page)

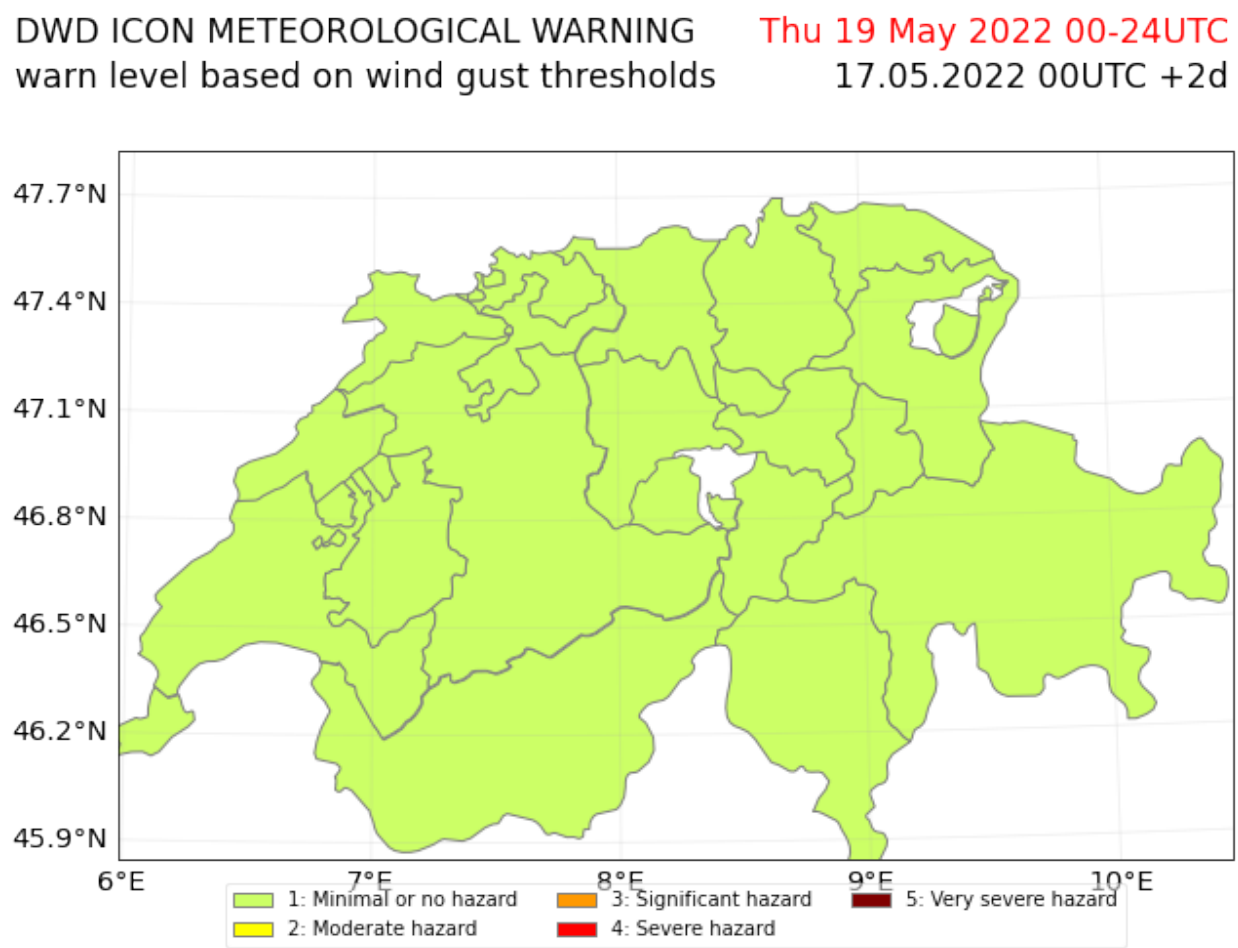
```
from climada.entity.exposures import Exposures
from climada.entity.impact_funcs import ImpactFunc, ImpactFuncSet
import climada.util.plot as u_plot

### generate exposure
# find out which hazard coord to consider
CHE_borders = u_plot._get_borders(
    np.stack([exposure.latitude, exposure.longitude], axis=1)
)
centroid_selection = np.logical_and(
    np.logical_and(
        hazard.centroids.lat >= CHE_borders[2], hazard.centroids.lat <= CHE_borders[3]
    ),
    np.logical_and(
        hazard.centroids.lon >= CHE_borders[0], hazard.centroids.lon <= CHE_borders[1]
    ),
)
# Fill DataFrame with values for a "neutral" exposure (value = 1)

exp_df = DataFrame()
exp_df["value"] = np.ones_like(
    hazard.centroids.lat[centroid_selection]
)  # provide value
exp_df["latitude"] = hazard.centroids.lat[centroid_selection]
exp_df["longitude"] = hazard.centroids.lon[centroid_selection]
exp_df["impf_WS"] = np.ones_like(hazard.centroids.lat[centroid_selection], int)
# Generate Exposures
exp = Exposures(exp_df)
exp.check()
exp.value_unit = "warn_level"

### generate impact functions
## impact functions for hazard based warnings
haz_type = "WS"
idx = 1
name = "warn_level_low_elevation"
intensity_unit = "m/s"
intensity = np.array(
    [0.0, 19.439, 19.44, 24.999, 25.0, 30.549, 30.55, 38.879, 38.88, 100.0]
)
mdd = np.array([1.0, 1.0, 2.0, 2.0, 3.0, 3.0, 4.0, 4.0, 5.0, 5.0])
paa = np.ones_like(mdd)
imp_fun_low = ImpactFunc(haz_type, idx, intensity, mdd, paa, intensity_unit, name)
imp_fun_low.check()
# fill ImpactFuncSet
impf_set = ImpactFuncSet([imp_fun_low])
```

```
2022-05-17 09:07:44,196 - climada.entity.impact_funcs.base - WARNING - For intensity␣
↪= 0, mdd != 0 or paa != 0. Consider shifting the origin of the intensity scale. In␣
↪impact.calc the impact is always null at intensity = 0.
```

```
# create and calculate Forecast
warn_forecast = Forecast({run_datetime: hazard}, exp, impf_set)
warn_forecast.calc()
```

The each grid point now has a warnlevel between 1-5 assigned for each event. Now the cantons can be colored based on a threshold on a grid point level. for each warning level it is assessed if 50% of grid points in the area of a canton has at least a 50% probability of reaching the specified threshold.

```
warn_forecast.plot_warn_map(
    cantons_file,
    thresholds=[2, 3, 4, 5],
    decision_level="exposure_point",
    probability_aggregation=0.5,
    area_aggregation=0.5,
    title="DWD ICON METEOROLOGICAL WARNING",
    explain_text="warn level based on wind gust thresholds",
    save_fig=False,
    close_fig=False,
    proj=ccrs.epsg(2056),
);
```



2.8.3 Example: Forecast of Tropical Cylcones' Tracks, Windfields and Impacts

The TCForecast class which facilitates fetching tropical cyclone forecasts from ECMWF is located in climada-petals. In the Weather4UN pilot project at MeteoSwiss an approach to calculate global impact estimates of tropical cyclones has been developed. It is documented in this github repository: https://github.com/ThomasRoosli/Weather4UN_global_impact_estimates. If you are interested in running an example computation, follow the steps described in the README.

2.9 Impact Function Calibration

CLIMADA provides the `climada.util.calibrate` module for calibrating impact functions based on impact data. This tutorial will guide through the usage of this module by calibrating an impact function for tropical cyclones (TCs).

For further information on the classes available from the module, see its *documentation*.

2.9.1 Overview

The basic idea of the calibration is to find a set of parameters for an impact function that minimizes the deviation between the calculated impact and some impact data. For setting up a calibration task, users have to supply the following information:

- Hazard and Exposure (as usual, see *the tutorial*)
- The impact data to calibrate the model to
- An impact function definition depending on the calibrated parameters
- Bounds and constraints of the calibrated parameters (depending on the calibration algorithm)
- A "cost function" defining the single-valued deviation between impact data and calculated impact
- A function for transforming the calculated impact into the same data structure as the impact data

This information defines the calibration task and is inserted into the `Input` object. Afterwards, the user may insert this object into one of the optimizer classes. Currently, the following classes are available:

- `BayesianOptimizer`: Uses Bayesian optimization to sample the parameter space.
- `ScipyMinimizeOptimizer`: Uses the `scipy.optimize.minimize` function for determining the best parameter set.

The following tutorial walks through the input data preparation and the setup of a `BayesianOptimizer` instance for calibration. For a brief example, refer to *Quickstart*. If you want to go through a somewhat realistic calibration task step-by-step, continue with *Calibration Data*.

```
import logging
import climada

logging.getLogger("climada").setLevel("WARNING")
```

Quickstart

This section gives a very quick overview of assembling a calibration task. Here, we calibrate a single impact function for damage reports in Mexico (`MEX`) from a TC with IbtracsID `2010176N16278`.

```
import pandas as pd

from climada.util import log_level
```

(continues on next page)

(continued from previous page)

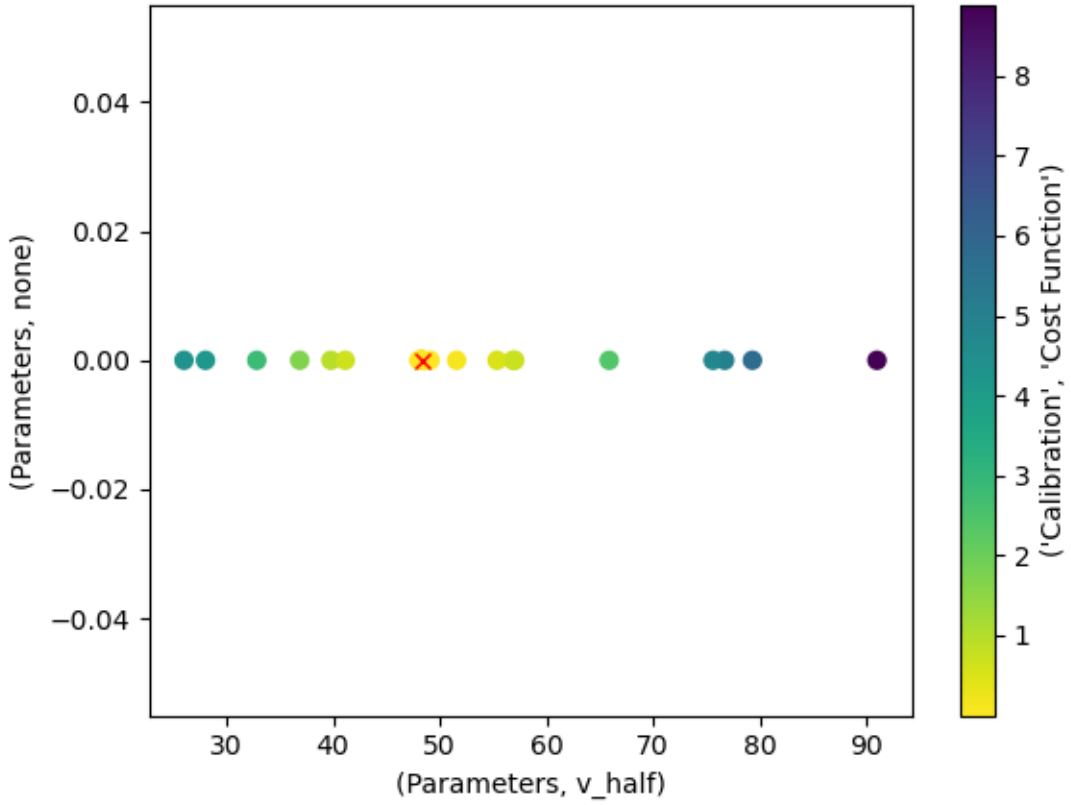
```
from climada.util.api_client import Client
from climada.entity import ImpactFuncSet, ImpfTropCyclone
from climada.util.calibrate import (
    Input,
    BayesianOptimizer,
    BayesianOptimizerController,
    OutputEvaluator,
    msle,
)

# Load hazard and exposure from Data API
client = Client()
exposure = client.get_litpop("MEX")
exposure.gdf["impf_TC"] = 1
all_tcs = client.get_hazard(
    "tropical_cyclone",
    properties={"event_type": "observed", "spatial_coverage": "global"},
)
hazard = all_tcs.select(event_names=["2010176N16278"])

# Impact data (columns: region ID, index: hazard event ID)
data = pd.DataFrame(data=[[2.485465e09]], columns=[484], index=list(hazard.event_id))

# Create input
inp = Input(
    hazard=hazard,
    exposure=exposure,
    data=data,
    # Generate impact function from estimated parameters
    impact_func_creator=lambda v_half: ImpactFuncSet(
        [ImpfTropCyclone.from_emanuel_usa(v_half=v_half, impf_id=1)]
    ),
    # Estimated parameter bounds
    bounds={"v_half": (26, 100)},
    # Cost function
    cost_func=msle,
    # Transform impact to pandas Dataframe with same structure as data
    impact_to_dataframe=lambda impact: impact.impact_at_reg(exposure.gdf["region_id
↪"]),
)

# Set up optimizer (with controller)
controller = BayesianOptimizerController.from_input(inp)
opt = BayesianOptimizer(inp)

# Run optimization
with log_level("WARNING", "climada.engine.impact_calc"):
    output = opt.run(controller=controller)

# Analyse results
output.plot_p_space()
out_eval = OutputEvaluator(inp, output)
```

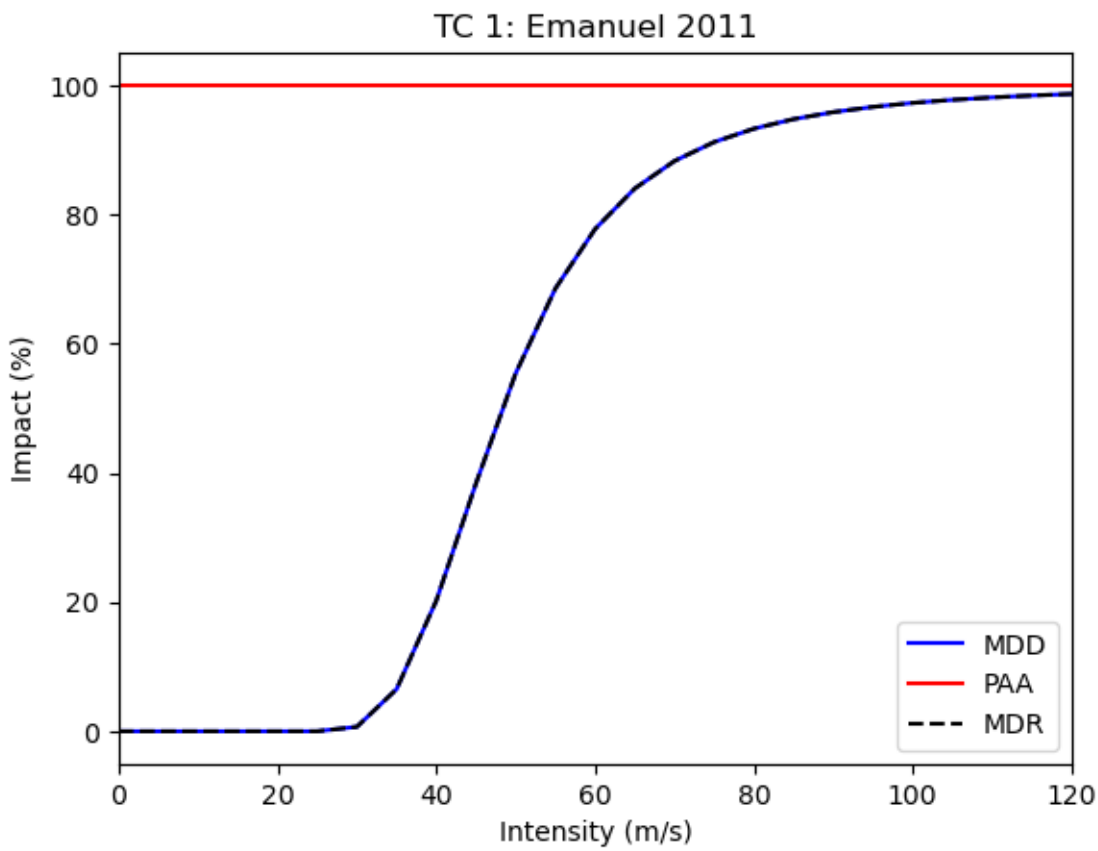
(continues on next page)

(continued from previous page)

```
out_eval.impf_set.plot()

# Optimal value
output.params
```

```
{'v_half': 48.30330871103452}
```





Follow the next sections of the tutorial for a more in-depth explanation.

2.9.2 Calibration Data

CLIMADA ships data from the International Disaster Database EM-DAT, which we will use to calibrate impact functions on. In the first step, we will select TC events that caused damages in the `NA1` basin since 2010.

We use EMDAT data from TCs occurring in the `NA1` basin since 2010. We calculate the centroids for which we want to compute the windfields by extracting the countries hit by the cyclones and retrieving a `LitPop` exposure instance from them. We then use the exposure coordinates as centroids coordinates.

```
import pandas as pd
from climada.util.constants import SYSTEM_DIR

emdat = pd.read_csv(SYSTEM_DIR / "tc_impf_cal_v01_EDR.csv")
emdat_subset = emdat[(emdat["cal_region2"] == "NA1") & (emdat["year"] >= 2010)]
emdat_subset
```

```
    country  region_id cal_region2  year      EM_ID      ibtracsID  \
326     MEX        484         NA1  2010  2010-0260  2010176N16278
331     ATG         28         NA1  2010  2010-0468  2010236N12341
334     MEX        484         NA1  2010  2010-0494  2010257N16282
339     LCA        662         NA1  2010  2010-0571  2010302N09306
340     VCT        670         NA1  2010  2010-0571  2010302N09306
344     BHS         44         NA1  2011  2011-0328  2011233N15301
345     DOM        214         NA1  2011  2011-0328  2011233N15301
```

(continues on next page)

(continued from previous page)

```
346     PRI        630         NA1  2011  2011-0328  2011233N15301
352     MEX        484         NA1  2011  2011-0385  2011279N10257
359     MEX        484         NA1  2012  2012-0276  2012215N12313
365     MEX        484         NA1  2012  2012-0401  2012166N09269
369     JAM        388         NA1  2012  2012-0410  2012296N14283
406     MEX        484         NA1  2014  2014-0333  2014253N13260
427     MEX        484         NA1  2015  2015-0470  2015293N13266
428     CPV        132         NA1  2015  2015-0473  2015242N12343
429     BHS         44         NA1  2015  2015-0479  2015270N27291
437     MEX        484         NA1  2016  2016-0319  2016248N15255
451     MEX        484         NA1  2017  2017-0334  2017219N16279
456     ATG         28         NA1  2017  2017-0381  2017242N16333
457     BHS         44         NA1  2017  2017-0381  2017242N16333
458     CUB        192         NA1  2017  2017-0381  2017242N16333
459     KNA        659         NA1  2017  2017-0381  2017242N16333
460     TCA        796         NA1  2017  2017-0381  2017242N16333
462     VGB         92         NA1  2017  2017-0381  2017242N16333
463     DMA        212         NA1  2017  2017-0383  2017260N12310
464     DOM        214         NA1  2017  2017-0383  2017260N12310
465     PRI        630         NA1  2017  2017-0383  2017260N12310

     emdat_impact  reference_year  emdat_impact_scaled  climada_impact  ...  \
326  2.000000e+09            2014         2.485465e+09    2.478270e+09  ...
331  1.260000e+07            2014         1.394594e+07    1.402875e+07  ...
334  3.900000e+09            2014         4.846656e+09    4.857140e+09  ...
339  5.000000e+05            2014         5.486675e+05    5.492871e+05  ...
340  2.500000e+07            2014         2.670606e+07    2.676927e+07  ...
344  4.000000e+07            2014         4.352258e+07    4.339898e+07  ...
345  3.000000e+07            2014         3.428317e+07    3.404744e+07  ...
346  5.000000e+08            2014         5.104338e+08    5.139659e+08  ...
352  2.770000e+07            2014         3.084603e+07    3.077374e+07  ...
359  3.000000e+08            2014         3.283428e+08    3.284805e+08  ...
365  5.550000e+08            2014         6.074341e+08    6.103254e+08  ...
369  1.654200e+07            2014         1.548398e+07    1.548091e+07  ...
406  2.500000e+09            2014         2.500000e+09    2.501602e+09  ...
427  8.230000e+08            2014         9.242430e+08    9.239007e+08  ...
428  1.100000e+06            2014         1.281242e+06    1.282842e+06  ...
429  9.000000e+07            2014         8.362720e+07    1.129343e+06  ...
437  5.000000e+07            2014         6.098482e+07    6.088808e+07  ...
451  2.000000e+06            2014         2.284435e+06    2.285643e+06  ...
456  2.500000e+08            2014         2.111764e+08    2.110753e+08  ...
457  2.000000e+06            2014         1.801876e+06    6.118379e+05  ...
458  1.320000e+10            2014         1.099275e+10    1.090424e+10  ...
459  2.000000e+07            2014         1.848489e+07    1.848665e+07  ...
460  5.000000e+08            2014         5.000000e+08    5.003790e+08  ...
462  3.000000e+09            2014         3.000000e+09    6.236200e+08  ...
463  1.456000e+09            2014         1.534596e+09    8.951186e+08  ...
464  6.300000e+07            2014         5.481371e+07    5.493466e+07  ...
465  6.800000e+10            2014         6.700905e+10    6.702718e+10  ...

     scale  log_ratio     unique_ID  Associated_disaster  Surge   Rain  Flood  \
326    1.0  -0.002899  2010-0260MEX                 True  False  False   True
```

(continues on next page)

(continued from previous page)

```
331    1.0   0.005920  2010-0468ATG                True  False  False   True
334    1.0   0.002161  2010-0494MEX                True  False  False   True
339    1.0   0.001129  2010-0571LCA                True  False  False   True
340    1.0   0.002364  2010-0571VCT               False  False  False  False
344    1.0  -0.002844  2011-0328BHS               False  False  False  False
345    1.0  -0.006900  2011-0328DOM                True  False  False   True
346    1.0   0.006896  2011-0328PRI                True  False  False   True
352    1.0  -0.002346  2011-0385MEX                True  False  False   True
359    1.0   0.000419  2012-0276MEX               False  False  False  False
365    1.0   0.004749  2012-0401MEX               False  False  False  False
369    1.0  -0.000198  2012-0410JAM               False  False  False  False
406    1.0   0.000640  2014-0333MEX                True  False  False   True
427    1.0  -0.000370  2015-0470MEX                True  False  False   True
428    1.0   0.001248  2015-0473CPV               False  False  False  False
429    1.0  -4.304733  2015-0479BHS                True   True  False   True
437    1.0  -0.001588  2016-0319MEX                True  False  False   True
451    1.0   0.000529  2017-0334MEX                True  False  False   True
456    1.0  -0.000479  2017-0381ATG               False  False  False  False
457    1.0  -1.080116  2017-0381BHS               False  False  False  False
458    1.0  -0.008085  2017-0381CUB                True  False  False   True
459    1.0   0.000095  2017-0381KNA               False  False  False  False
460    1.0   0.000758  2017-0381TCA                True  False  False   True
462    1.0  -1.570826  2017-0381VGB               False  False  False  False
463    1.0  -0.539066  2017-0383DMA                True  False  False   True
464    1.0   0.002204  2017-0383DOM                True  False  False   True
465    1.0   0.000271  2017-0383PRI                True  False  False   True

     Slide  Other  OtherThanSurge
326  False  False            True
331  False  False            True
334  False  False            True
339   True  False            True
340  False  False           False
344  False  False           False
345  False  False            True
346  False  False            True
352   True  False            True
359  False  False           False
365  False  False           False
369  False  False           False
406  False  False            True
427  False  False            True
428  False  False           False
429  False  False            True
437  False  False            True
451   True  False            True
456  False  False           False
457  False  False           False
458  False  False            True
459  False  False           False
460  False  False            True
462  False  False           False
```

(continues on next page)

(continued from previous page)

```
463   True  False            True
464   True  False            True
465  False   True            True

[27 rows x 22 columns]
```

Each entry in the database refers to an economic impact for a specific country and TC event. The TC events are identified by the ID assigned from the International Best Track Archive for Climate Stewardship (IBTrACS). We now want to reshape this data so that impacts are grouped by event and country.

To achieve this, we iterate over the unique track IDs, select all reported damages associated with this ID, and concatenate the results. For missing entries, `pandas` will set the value to NaN. We assume that missing entries means that no damages are reported (this is a strong assumption), and set all NaN values to zero. Then, we transpose the dataframe so that each row represents an event and each column states the damage for a specific country. Finally, we set the track ID to be the index of the data frame.

```
track_ids = emdat_subset["ibtracsID"].unique()

data = pd.pivot_table(
    emdat_subset,
    values="emdat_impact_scaled",
    index="ibtracsID",
    columns="region_id",
    # fill_value=0,
)
data
```

```
region_id               28            44            92           132  \
ibtracsID
2010176N16278           NaN           NaN           NaN          NaN
2010236N12341  1.394594e+07           NaN           NaN          NaN
2010257N16282           NaN           NaN           NaN          NaN
2010302N09306           NaN           NaN           NaN          NaN
2011233N15301           NaN  4.352258e+07           NaN          NaN
2011279N10257           NaN           NaN           NaN          NaN
2012166N09269           NaN           NaN           NaN          NaN
2012215N12313           NaN           NaN           NaN          NaN
2012296N14283           NaN           NaN           NaN          NaN
2014253N13260           NaN           NaN           NaN          NaN
2015242N12343           NaN           NaN           NaN  1281242.483
2015270N27291           NaN  8.362720e+07           NaN          NaN
2015293N13266           NaN           NaN           NaN          NaN
2016248N15255           NaN           NaN           NaN          NaN
2017219N16279           NaN           NaN           NaN          NaN
2017242N16333  2.111764e+08  1.801876e+06  3.000000e+09          NaN
2017260N12310           NaN           NaN           NaN          NaN

region_id               192           212          214          388  \
ibtracsID
2010176N16278           NaN           NaN          NaN          NaN
2010236N12341           NaN           NaN          NaN          NaN
2010257N16282           NaN           NaN          NaN          NaN
```

(continues on next page)

(continued from previous page)

```
2010302N09306           NaN           NaN          NaN          NaN
2011233N15301           NaN           NaN  34283168.75          NaN
2011279N10257           NaN           NaN          NaN          NaN
2012166N09269           NaN           NaN          NaN          NaN
2012215N12313           NaN           NaN          NaN          NaN
2012296N14283           NaN           NaN          NaN  15483975.86
2014253N13260           NaN           NaN          NaN          NaN
2015242N12343           NaN           NaN          NaN          NaN
2015270N27291           NaN           NaN          NaN          NaN
2015293N13266           NaN           NaN          NaN          NaN
2016248N15255           NaN           NaN          NaN          NaN
2017219N16279           NaN           NaN          NaN          NaN
2017242N16333  1.099275e+10           NaN          NaN          NaN
2017260N12310           NaN  1.534596e+09  54813712.03          NaN

region_id               484           630          659          662  \
ibtracsID
2010176N16278  2.485465e+09           NaN          NaN          NaN
2010236N12341           NaN           NaN          NaN          NaN
2010257N16282  4.846656e+09           NaN          NaN          NaN
2010302N09306           NaN           NaN          NaN  548667.5019
2011233N15301           NaN  5.104338e+08          NaN          NaN
2011279N10257  3.084603e+07           NaN          NaN          NaN
2012166N09269  6.074341e+08           NaN          NaN          NaN
2012215N12313  3.283428e+08           NaN          NaN          NaN
2012296N14283           NaN           NaN          NaN          NaN
2014253N13260  2.500000e+09           NaN          NaN          NaN
2015242N12343           NaN           NaN          NaN          NaN
2015270N27291           NaN           NaN          NaN          NaN
2015293N13266  9.242430e+08           NaN          NaN          NaN
2016248N15255  6.098482e+07           NaN          NaN          NaN
2017219N16279  2.284435e+06           NaN          NaN          NaN
2017242N16333           NaN           NaN  18484889.46          NaN
2017260N12310           NaN  6.700905e+10          NaN          NaN

region_id              670          796
ibtracsID
2010176N16278          NaN          NaN
2010236N12341          NaN          NaN
2010257N16282          NaN          NaN
2010302N09306  26706058.15          NaN
2011233N15301          NaN          NaN
2011279N10257          NaN          NaN
2012166N09269          NaN          NaN
2012215N12313          NaN          NaN
2012296N14283          NaN          NaN
2014253N13260          NaN          NaN
2015242N12343          NaN          NaN
2015270N27291          NaN          NaN
2015293N13266          NaN          NaN
2016248N15255          NaN          NaN
2017219N16279          NaN          NaN
```

(continues on next page)

(continued from previous page)

```
2017242N16333          NaN  500000000.0
2017260N12310          NaN          NaN
```

This is the data against which we want to compare our model output. Let's continue setting up the calibration!

2.9.3 Model Setup

In the first step, we create the exposure layer for the model. We use the `LitPop` module and simply pass the names of all countries listed in our calibration data to the `from_countries()` classmethod. The countries are the columns in the `data` object.

Alternatively, we could have inserted `emdat_subset["region_id"].unique().tolist()`.

```
# from climada.entity.exposures.litpop import LitPop
# from climada.util import log_level

# # Calculate the exposure
# with log_level("ERROR"):
#     exposure = LitPop.from_countries(data.columns.tolist())
```

```
from climada.util.api_client import Client
from climada.entity.exposures.litpop import LitPop
from climada.util.coordinates import country_to_iso

client = Client()
exposure = LitPop.concat(
    [
        client.get_litpop(country_to_iso(country_id, representation="alpha3"))
        for country_id in data.columns
    ]
)
```

```
from climada.util.api_client import Client

client = Client()
tc_dataset_infos = client.list_dataset_infos(data_type="tropical_cyclone")
client.get_property_values(
    client.list_dataset_infos(data_type="tropical_cyclone"),
    known_property_values={"event_type": "observed", "spatial_coverage": "global"},
)
```

```
{'res_arcsec': ['150'],
 'event_type': ['observed'],
 'spatial_coverage': ['global'],
 'climate_scenario': ['None']}
```

We will use the CLIMADA Data API to download readily computed wind fields from TC tracks. The API provides a large dataset containing all historical TC tracks. We will download them and then select the subset of TCs for which we have impact data by using `select()`.

```
from climada.util.api_client import Client
```

(continues on next page)

(continued from previous page)

```
client = Client()
all_tcs = client.get_hazard(
    "tropical_cyclone",
    properties={"event_type": "observed", "spatial_coverage": "global"},
)
hazard = all_tcs.select(event_names=track_ids.tolist())
```

NOTE: Discouraged! This will usually take a longer time than using the Data API

Alternatively, CLIMADA provides the *TCTracks* class, which lets us download the tracks of TCs using their IBTrACS IDs. We then have to equalize the time steps of the different TC tracks.

The track and intensity of a cyclone are insufficient to compute impacts in CLIMADA. We first have to re-compute a windfield from each track at the locations of interest. For consistency, we simply choose the coordinates of the exposure.

```
# NOTE: Uncomment this to compute wind fields yourself

# from climada.hazard import Centroids, TCTracks, TropCyclone

# # Get the tracks for associated TCs
# tracks = TCTracks.from_ibtracs_netcdf(storm_id=track_ids.tolist())
# tracks.equal_timestep(time_step_h=1.0, land_params=False)
# tracks.plot()

# # Calculate windfield for the tracks
# centroids = Centroids.from_lat_lon(exposure.gdf['latitude'], exposure.gdf['longitude
↪'])
# hazard = TropCyclone.from_tracks(tracks, centroids)
```

2.9.4 Calibration Setup

We are now set up to define the specifics of the calibration task. First, let us define the impact function we actually want to calibrate. We select the formula by Emanuel (2011), for which a shortcut exists in the CLIMADA code base: *from_emanuel_usa()*. The sigmoid-like impact function takes three parameters, the wind threshold for any impact `v_thresh`, the wind speed where half of the impact occurs `v_half`, and the maximum impact factor `scale`. According to the model by Emanuel (2011), `v_thresh` is considered a constant, so we choose to only calibrate the latter two.

Any CLIMADA *Optimizer* will roughly perform the following algorithm:

1. Create a set of parameters and built an impact function (set) from it.
2. Compute an impact with that impact function set.
3. Compare the impact and the calibration data via the cost/target function.
4. Repeat N times or until the target function goal is reached.

The selection of parameters is based on the target function varies strongly between different optimization algorithms.

For the first step, we have to supply a function that takes the parameters we try to estimate and returns the impact function set that can later be used in an impact calculation. We only calibrate a single function for the entire basin, so this is straightforward.

To ensure the impact function is applied correctly, we also have to set the `impf_` column of the exposure `GeoDataFrame`. Note that the default impact function ID is 1, and that the hazard type is `"TC"`.

```
from climada.entity import ImpactFuncSet, ImpfTropCyclone

# Match impact function and exposure
exposure.gdf["impf_TC"] = 1


def impact_func_tc(v_half, scale):
    return ImpactFuncSet([ImpfTropCyclone.from_emanuel_usa(v_half=v_half,␣
↪scale=scale)])
```

We will be using the *BayesianOptimizer*, which requires very little information on the parameter space beforehand. One crucial information are the bounds of the parameters, though. Initial values are not needed because the optimizer first samples the bound parameter space uniformly and then iteratively "narrows down" the search. We choose a `v_half` between `v_thresh` and 150, and a scale between 0.01 (it must never be zero) and 1.0. Specifying the bounds as dictionary (a must in case of `BayesianOptimizer`) also serves the purpose of naming the parameters we want to calibrate. Notice that these names have to match the arguments of the impact function generator.

```
bounds = {"v_half": (25.8, 150), "scale": (0.01, 1)}
```

Defining the cost function is crucial for the result of the calibration. You can choose what is best suited for your application. Often, it is not clear which function works best, and it's a good idea to try out a few. Because the impacts of different events may vary over several orders of magnitude, we select the mean squared logartihmic error (MSLE). This one and other error measures are readily supplied by the `sklearn` package.

The cost function must be defined as a function that takes the impact object calculated by the optimization algorithm and the input calibration data as arguments, and that returns a single number. This number represents a "cost" of the parameter set used for calculating the impact. A higher cost therefore is worse, a lower cost is better. Any optimizer will try to minimize the cost.

Note that the impact object is an instance of `Impact`, whereas the input calibration data is a `pd.DataFrame`. To compute the MSLE, we first have to transform the impact into the same data structure, meaning that we have to aggregate the point-wise impacts by event and country. The function performing this transformation task is provided to the *Input* via its *impact_to_dataframe* attribute. Here we choose *climada.engine.impact.Impact.impact_at_reg()*, which aggregates over countries by default. To improve performance, we can supply this function with our known region IDs instead of re-computing them in every step.

Computations on data frames align columns and indexes. The indexes of the calibration data are the IBTrACS IDs, but the indexes of the result of `impact_at_reg` are the hazard event IDs, which at this point are only integer numbers. To resolve that, we adjust our calibration dataframe to carry the respective `Hazard.event_id` as index.

```
data = data.rename(
    index={
        hazard.event_name[idx]: hazard.event_id[idx]
        for idx in range(len(hazard.event_id))
    }
)
data.index.rename("event_id", inplace=True)
data
```

```
region_id           28            44            92           132  \
event_id
1333                NaN           NaN           NaN           NaN
1339       1.394594e+07           NaN           NaN           NaN
1344                NaN           NaN           NaN           NaN
```

(continues on next page)

(continued from previous page)

```
1351                NaN           NaN           NaN           NaN
1361                NaN  4.352258e+07           NaN           NaN
3686                NaN           NaN           NaN           NaN
3691                NaN           NaN           NaN           NaN
1377                NaN           NaN           NaN           NaN
1390                NaN           NaN           NaN           NaN
3743                NaN           NaN           NaN           NaN
1421                NaN           NaN           NaN  1281242.483
1426                NaN  8.362720e+07           NaN           NaN
3777                NaN           NaN           NaN           NaN
3795                NaN           NaN           NaN           NaN
1450                NaN           NaN           NaN           NaN
1454       2.111764e+08  1.801876e+06  3.000000e+09           NaN
1458                NaN           NaN           NaN           NaN

region_id           192           212          214          388           484  \
event_id
1333                NaN           NaN          NaN          NaN  2.485465e+09
1339                NaN           NaN          NaN          NaN           NaN
1344                NaN           NaN          NaN          NaN  4.846656e+09
1351                NaN           NaN          NaN          NaN           NaN
1361                NaN           NaN  34283168.75          NaN           NaN
3686                NaN           NaN          NaN          NaN  3.084603e+07
3691                NaN           NaN          NaN          NaN  6.074341e+08
1377                NaN           NaN          NaN          NaN  3.283428e+08
1390                NaN           NaN          NaN  15483975.86           NaN
3743                NaN           NaN          NaN          NaN  2.500000e+09
1421                NaN           NaN          NaN          NaN           NaN
1426                NaN           NaN          NaN          NaN           NaN
3777                NaN           NaN          NaN          NaN  9.242430e+08
3795                NaN           NaN          NaN          NaN  6.098482e+07
1450                NaN           NaN          NaN          NaN  2.284435e+06
1454       1.099275e+10           NaN          NaN          NaN           NaN
1458                NaN  1.534596e+09  54813712.03          NaN           NaN

region_id           630          659          662          670          796
event_id
1333                NaN          NaN          NaN          NaN          NaN
1339                NaN          NaN          NaN          NaN          NaN
1344                NaN          NaN          NaN          NaN          NaN
1351                NaN          NaN  548667.5019  26706058.15          NaN
1361       5.104338e+08          NaN          NaN          NaN          NaN
3686                NaN          NaN          NaN          NaN          NaN
3691                NaN          NaN          NaN          NaN          NaN
1377                NaN          NaN          NaN          NaN          NaN
1390                NaN          NaN          NaN          NaN          NaN
3743                NaN          NaN          NaN          NaN          NaN
1421                NaN          NaN          NaN          NaN          NaN
1426                NaN          NaN          NaN          NaN          NaN
3777                NaN          NaN          NaN          NaN          NaN
3795                NaN          NaN          NaN          NaN          NaN
1450                NaN          NaN          NaN          NaN          NaN
```

(continues on next page)

(continued from previous page)

```
1454                NaN  18484889.46          NaN          NaN  500000000.0
1458       6.700905e+10          NaN          NaN          NaN          NaN
```

2.9.5 Execute the Calibration

We created a class *BayesianOptimizerController* to control and guide the calibration process. It is intended to walk through several optimization iterations and stop the process if the best guess cannot be improved. The optimization works as follows:

1. The optimizer randomly samples the parameter space *init_points* times.
2. The optimizer uses a Gaussian regression process to "smartly" sample the parameter space at most *n_iter* times.
   - The process uses an "Upper Confidence Bound" sampling method whose parameter *kappa* indicates how close the sampled points are to the buest guess. Higher kappa means more exploration of the parameter space, lower kappa means more exploitation.
   - After each sample, the parameter kappa is reduced by the factor kappa_decay. By default, this parameter is set such that kappa equals *kappa_min* at the last step. This way, the sampling becomes more exploitative the more steps are taken.
3. The controller tracks the improvements of the buest guess for parameters. If *min_improvement_count* consecutive improvements are lower than *min_improvement*, the smart sampling is stopped. In this case, the iterations count is increased and the process repeated from step 1.
4. If an entire iteration did not show any improvement, the optimization is stopped. It is also stopped when the *max_iterations* count is reached.

Users can control the "density", and thus the accuracy of the sampling by adjusting the controller parameters. Increasing init_points, n_iter, min_improvement_count, and max_iterations, and decreasing min_improvement generally increases density and accuracy, but leads to longer runtimes.

We suggest using the from_input classmethod for a convenient choice of sampling density based on the parameter space. The two parameters init_points and n_iter are set to b^N, where N is the number of estimated parameters and b is the sampling_base parameter, which defaults to 4.

Now we can finally execute our calibration task! We will plug all input parameters in an instance of Input, and then create the optimizer instance with it. The Optimizer.run method returns an *Output* object, whose *params* attribute holds the optimal parameters determined by the calibration.

Notice that the BayesianOptimization *maximizes* a target function. Therefore, higher target values are *better* than lower ones in this case.

```
from climada.util.calibrate import (
    Input,
    BayesianOptimizer,
    BayesianOptimizerController,
    msle,
)

from climada.util import log_level

# Define calibration input
with log_level("INFO", name_prefix="climada.util.calibrate"):
    input = Input(
        hazard=hazard,
        exposure=exposure,
```

(continues on next page)

(continued from previous page)

```
        data=data,
        impact_func_creator=impact_func_tc,
        cost_func=msle,
        impact_to_dataframe=lambda imp: imp.impact_at_reg(exposure.gdf["region_id"]),
        bounds=bounds,
    )

    # Create and run the optimizer
    opt = BayesianOptimizer(input)
    controller = BayesianOptimizerController.from_input(input)
    bayes_output = opt.run(controller=controller)
    bayes_output.params  # The optimal parameters
```

```
2025-06-23 13:06:39,791 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 0
2025-06-23 13:06:42,001 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 1
2025-06-23 13:06:44,417 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 2
2025-06-23 13:06:46,862 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 3
2025-06-23 13:06:49,665 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 4
2025-06-23 13:06:52,649 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 5
2025-06-23 13:06:55,998 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 6
2025-06-23 13:06:59,703 - climada.util.calibrate.bayesian_optimizer - INFO - No␣
↪improvement. Stop optimization.
```

2.9.6 Evaluate Output

The Bayesian Optimizer returns the entire paramter space it sampled via `BayesianOptimizerOutput` We can find out a lot about the relation of the fitted parameters by investigating how the cost function value depends on them. We can retrieve the parameter space as `pandas.DataFrame` via `p_space_to_dataframe()`. This dataframe has MultiIndex columns. One group are the `Parameters`, the other holds information on the `Calibration` for each parameter set. Notice that the optimal parameter set is not necessarily the last entry in the parameter space!

```
p_space_df = bayes_output.p_space_to_dataframe()
p_space_df
```

```
          Parameters                Calibration
               scale      v_half Cost Function
Iteration
0           0.422852  115.264302      2.726950
1           0.010113   63.349706      4.133135
2           0.155288   37.268453      0.800611
3           0.194398   68.718642      1.683610
4           0.402800   92.721038      2.046407
...              ...         ...           ...
219         0.711724   51.676718      0.773888
```

(continues on next page)

(continued from previous page)

```
220         0.920147   48.767230      0.771611
221         0.903951   50.001947      0.765957
222         1.000000   52.968157      0.764202
223         0.799533   49.721016      0.765107

[224 rows x 3 columns]
```

In contrast, the controller only tracks the consecutive improvements of the best guess.

```
controller.improvements()
```

```
        iteration  random    target  improvement
sample
0               0    True -2.726950          inf
2               0    True -0.800611     2.406088
24              0   False -0.793780     0.008605
29              0   False -0.791065     0.003432
40              1   False -0.781115     0.012737
44              1   False -0.777910     0.004119
48              1   False -0.774136     0.004876
60              1   False -0.773844     0.000377
81              2   False -0.770928     0.003782
87              2   False -0.770386     0.000704
89              2   False -0.768256     0.002773
117             3   False -0.765309     0.003850
123             3   False -0.765110     0.000260
156             4   False -0.764939     0.000223
178             5   False -0.763951     0.001294
189             5   False -0.763910     0.000054
```

To get one group of the columns, simply access the item.

```
p_space_df["Parameters"]
```

```
              scale      v_half
Iteration
0          0.422852  115.264302
1          0.010113   63.349706
2          0.155288   37.268453
3          0.194398   68.718642
4          0.402800   92.721038
...             ...         ...
219        0.711724   51.676718
220        0.920147   48.767230
221        0.903951   50.001947
222        1.000000   52.968157
223        0.799533   49.721016

[224 rows x 2 columns]
```

Notice that the optimal parameter set is not necessarily the last entry in the parameter space! Therefore, let's order the parameter space by the ascending cost function values.

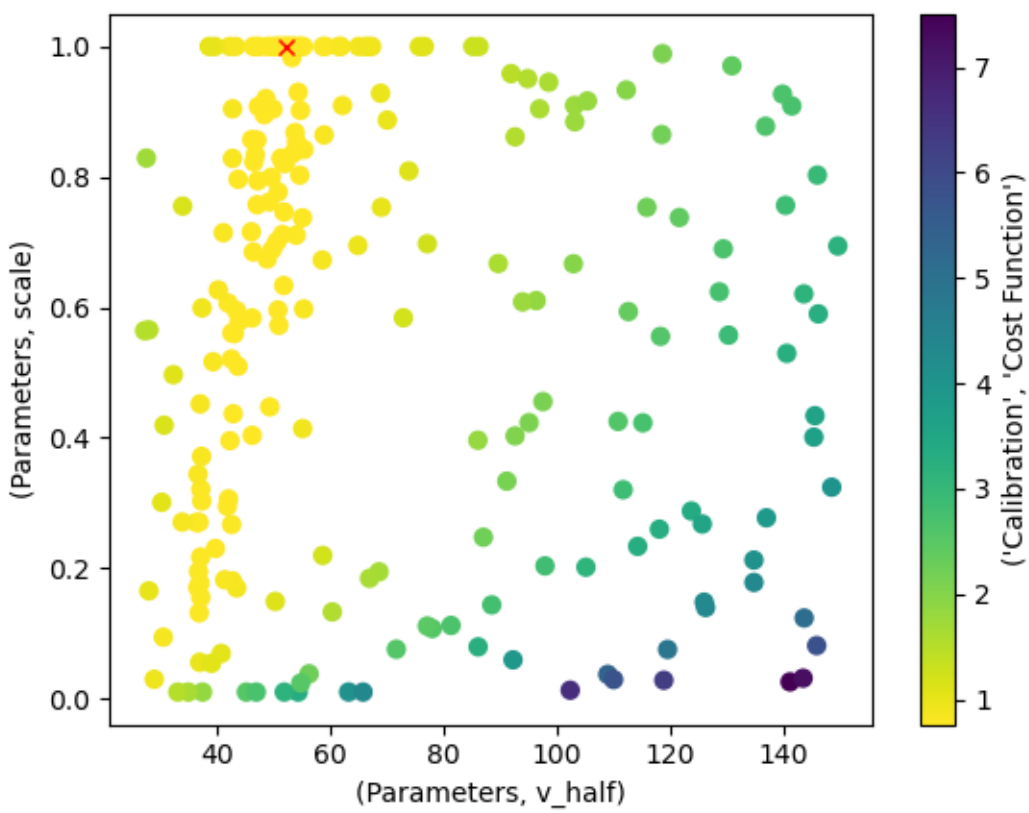
```
p_space_df = p_space_df.sort_values(("Calibration", "Cost Function"), ascending=True)
p_space_df
```

```
          Parameters                Calibration
               scale      v_half Cost Function
Iteration
189         1.000000   52.392753      0.763910
178         1.000000   52.176364      0.763951
185         1.000000   52.647234      0.763968
222         1.000000   52.968157      0.764202
180         1.000000   51.656629      0.764398
...              ...         ...           ...
139         0.081255  145.935720      5.822272
35          0.028105  118.967924      6.298332
97          0.012842  102.449398      6.635728
204         0.031310  143.537900      7.262185
101         0.025663  141.236104      7.504095

[224 rows x 3 columns]
```

The *BayesianOptimizerOutput* supplies the *plot_p_space()* method for convenience. If there were more than two parameters we calibrated, it would produce a plot for each parameter combination.

```
bayes_output.plot_p_space(x="v_half", y="scale")
```

```
<Axes: xlabel='(Parameters, v_half)', ylabel='(Parameters, scale)'>
```



```
p_space_df["Parameters"].iloc[0, :].to_dict()
```

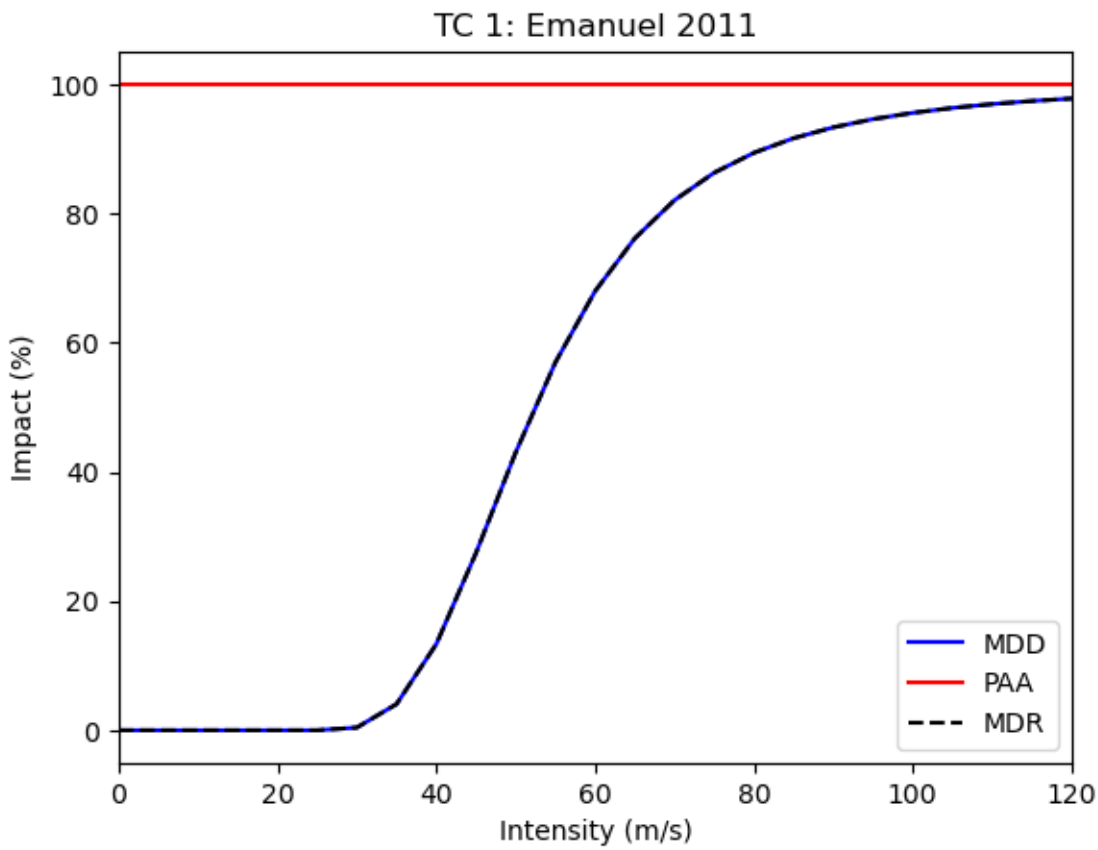
```
{'scale': 1.0, 'v_half': 52.392753384866985}
```

2.9.7 Analyze the Calibration

Now that we obtained a calibration result, we should investigate it further. The tasks of evaluating results and plotting them is simplified by the `OutputEvaluator`. It takes the input and output of a calibration task as parameters. Let's start by plotting the optimized impact function:

```
from climada.util.calibrate import OutputEvaluator

output_eval = OutputEvaluator(input, bayes_output)
output_eval.impf_set.plot()
```

```
<Axes: title={'center': 'TC 1: Emanuel 2011'}, xlabel='Intensity (m/s)', ylabel=
↪'Impact (%)'>
```



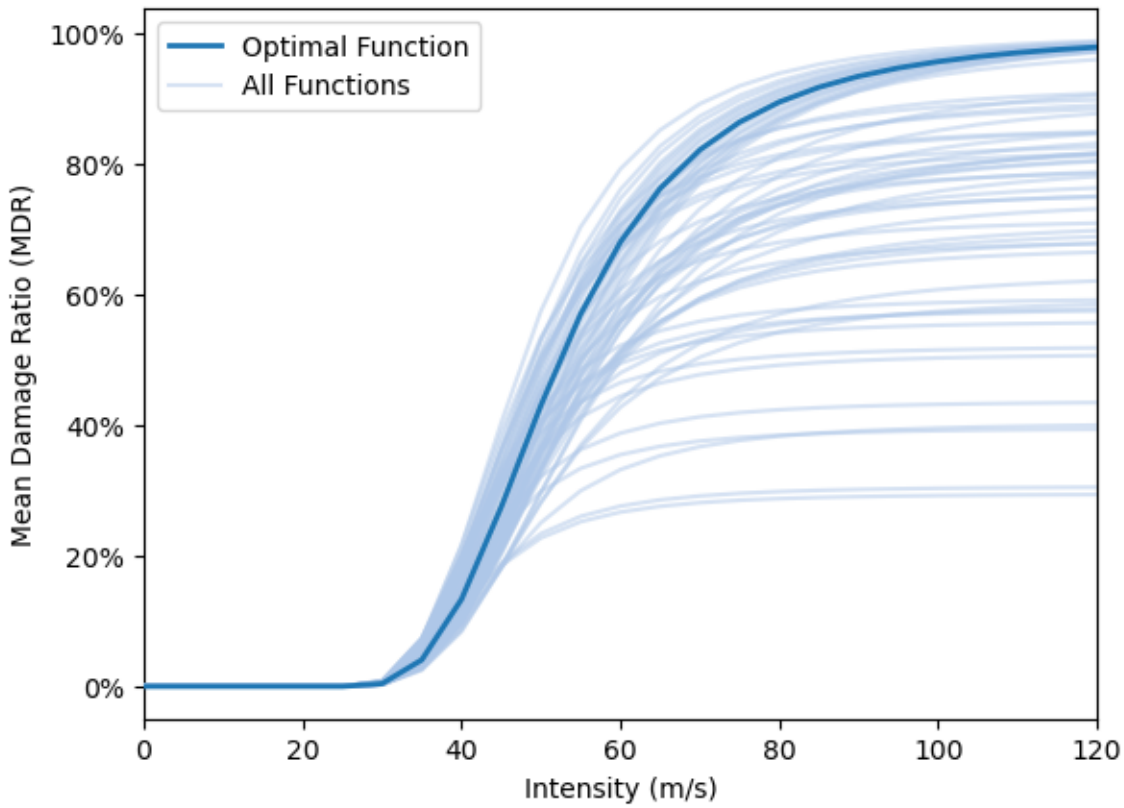
Here we show how the variability in parameter combinations with similar cost function values (as seen in the plot of the parameter space) translate to varying impact functions. In addition, the hazard value distribution is shown. Together this provides an intuitive overview regarding the robustness of the optimization, given the chosen cost function. It does NOT provide a view of the sampling uncertainty (as e.g. bootstrapping or cross-validation) NOR of the suitability of the cost function which is chosen by the user.

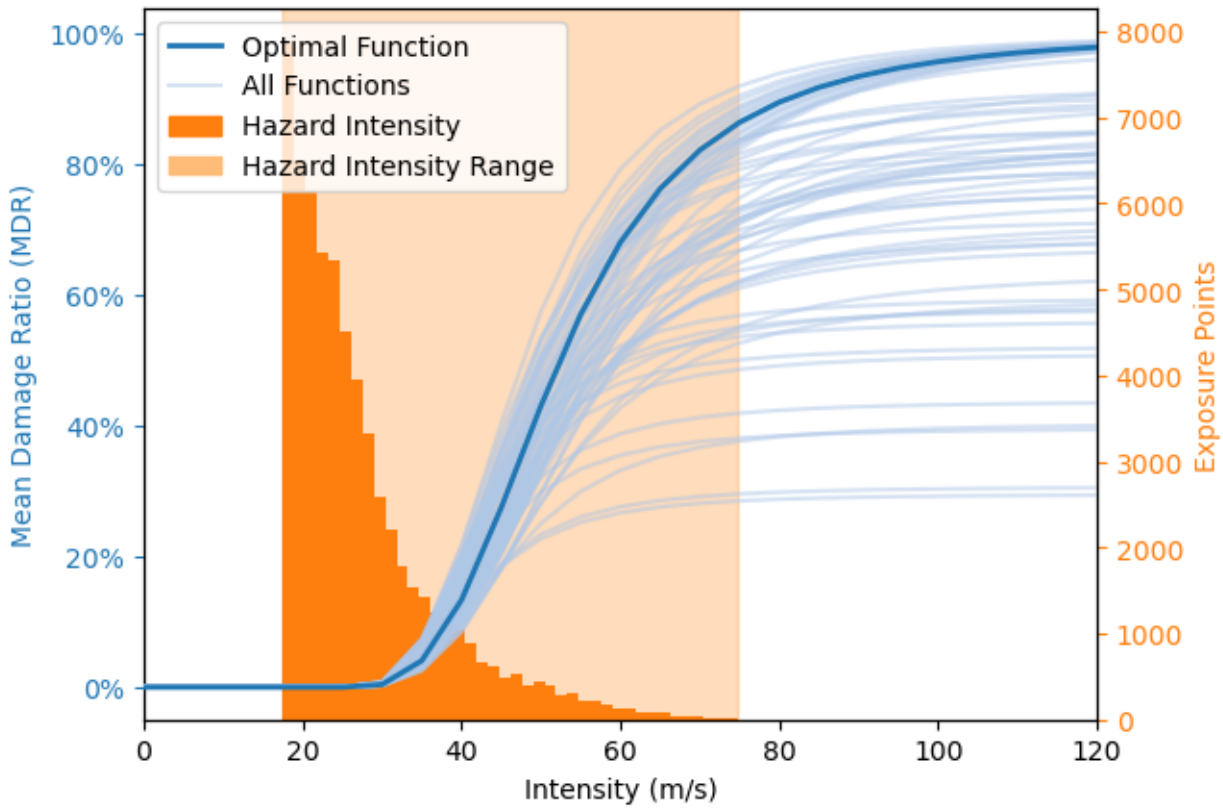
This functionality is only available from the `BayesianOptimizerOutputEvaluator` tailored to Bayesian optimizer

outputs. It includes all function from `OutputEvaluator`.

```
from climada.util.calibrate import BayesianOptimizerOutputEvaluator, select_best

output_eval = BayesianOptimizerOutputEvaluator(input, bayes_output)

# Plot the impact function variability
output_eval.plot_impf_variability(select_best(p_space_df, 0.03), plot_haz=False)
output_eval.plot_impf_variability(select_best(p_space_df, 0.03))
```

```
<Axes: xlabel='Intensity (m/s)', ylabel='Mean Damage Ratio (MDR)'>
```





The target function has limited meaning outside the calibration task. To investigate the quality of the calibration, it is helpful to compute the impact with the impact function defined by the optimal parameters. The `OutputEvaluator` readily computed this impact when it was created. You can access the impact via the `impact` attribute.

```
import numpy as np

impact_data = output_eval.impact.impact_at_reg(exposure.gdf["region_id"])
impact_data.set_index(np.asarray(hazard.event_name), inplace=True)
impact_data
```

```
                        28            44            92             132  \
2010176N16278  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2010236N12341  2.362421e+07  0.000000e+00  6.273895e+07       0.000000
2010257N16282  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2010302N09306  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2011233N15301  0.000000e+00  1.221974e+09  2.344844e+07       0.000000
2011279N10257  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2012215N12313  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2012166N09269  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2012296N14283  0.000000e+00  1.311881e+08  0.000000e+00       0.000000
2014253N13260  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2015293N13266  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2015242N12343  0.000000e+00  0.000000e+00  0.000000e+00  225671.780693
2015270N27291  0.000000e+00  8.412341e+05  0.000000e+00       0.000000
2016248N15255  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
2017219N16279  0.000000e+00  0.000000e+00  0.000000e+00       0.000000
```

(continues on next page)

(continued from previous page)

```
2017242N16333  3.732403e+08  1.348337e+06  4.696049e+08       0.000000
2017260N12310  0.000000e+00  0.000000e+00  3.385997e+07       0.000000

                        192           212           214           388  \
2010176N16278  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2010236N12341  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2010257N16282  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2010302N09306  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2011233N15301  0.000000e+00  0.000000e+00  7.044665e+04  0.000000e+00
2011279N10257  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2012215N12313  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2012166N09269  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2012296N14283  2.742997e+09  0.000000e+00  0.000000e+00  3.077018e+09
2014253N13260  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2015293N13266  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2015242N12343  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2015270N27291  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2016248N15255  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2017219N16279  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2017242N16333  3.351103e+09  0.000000e+00  2.670109e+07  0.000000e+00
2017260N12310  0.000000e+00  6.658893e+08  1.178713e+08  0.000000e+00

                        484           630           659           662  \
2010176N16278  1.629343e+09  0.000000e+00  0.000000e+00  0.000000e+00
2010236N12341  0.000000e+00  7.489799e+07  1.243066e+07  0.000000e+00
2010257N16282  9.210669e+07  0.000000e+00  0.000000e+00  0.000000e+00
2010302N09306  0.000000e+00  0.000000e+00  0.000000e+00  6.668112e+06
2011233N15301  0.000000e+00  7.033207e+08  0.000000e+00  0.000000e+00
2011279N10257  3.687471e+06  0.000000e+00  0.000000e+00  0.000000e+00
2012215N12313  1.564085e+08  0.000000e+00  0.000000e+00  0.000000e+00
2012166N09269  1.890366e+08  0.000000e+00  0.000000e+00  0.000000e+00
2012296N14283  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2014253N13260  1.353240e+10  0.000000e+00  0.000000e+00  0.000000e+00
2015293N13266  9.395908e+08  0.000000e+00  0.000000e+00  0.000000e+00
2015242N12343  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2015270N27291  0.000000e+00  0.000000e+00  0.000000e+00  0.000000e+00
2016248N15255  3.107738e+09  0.000000e+00  0.000000e+00  0.000000e+00
2017219N16279  9.019579e+07  0.000000e+00  0.000000e+00  0.000000e+00
2017242N16333  0.000000e+00  1.164926e+10  4.024997e+08  0.000000e+00
2017260N12310  0.000000e+00  8.941185e+10  0.000000e+00  0.000000e+00

                        670           796
2010176N16278  0.000000e+00  0.000000e+00
2010236N12341  0.000000e+00  0.000000e+00
2010257N16282  0.000000e+00  0.000000e+00
2010302N09306  4.537917e+06  6.931670e+05
2011233N15301  0.000000e+00  1.374375e+08
2011279N10257  0.000000e+00  0.000000e+00
2012215N12313  0.000000e+00  0.000000e+00
2012166N09269  0.000000e+00  0.000000e+00
2012296N14283  0.000000e+00  0.000000e+00
2014253N13260  0.000000e+00  0.000000e+00
```

(continues on next page)

(continued from previous page)

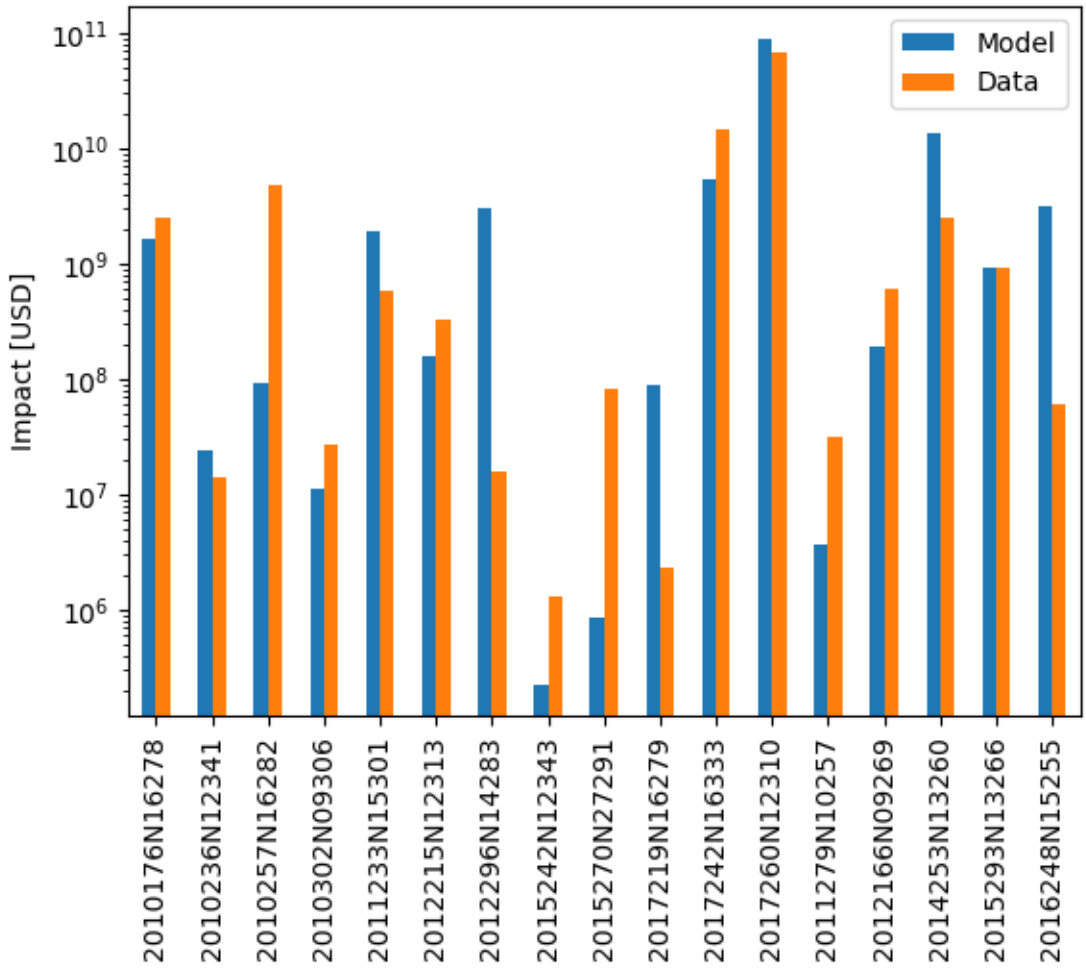
```
2015293N13266  0.000000e+00  0.000000e+00
2015242N12343  0.000000e+00  0.000000e+00
2015270N27291  0.000000e+00  0.000000e+00
2016248N15255  0.000000e+00  0.000000e+00
2017219N16279  0.000000e+00  0.000000e+00
2017242N16333  0.000000e+00  8.060703e+08
2017260N12310  0.000000e+00  9.182423e+07
```

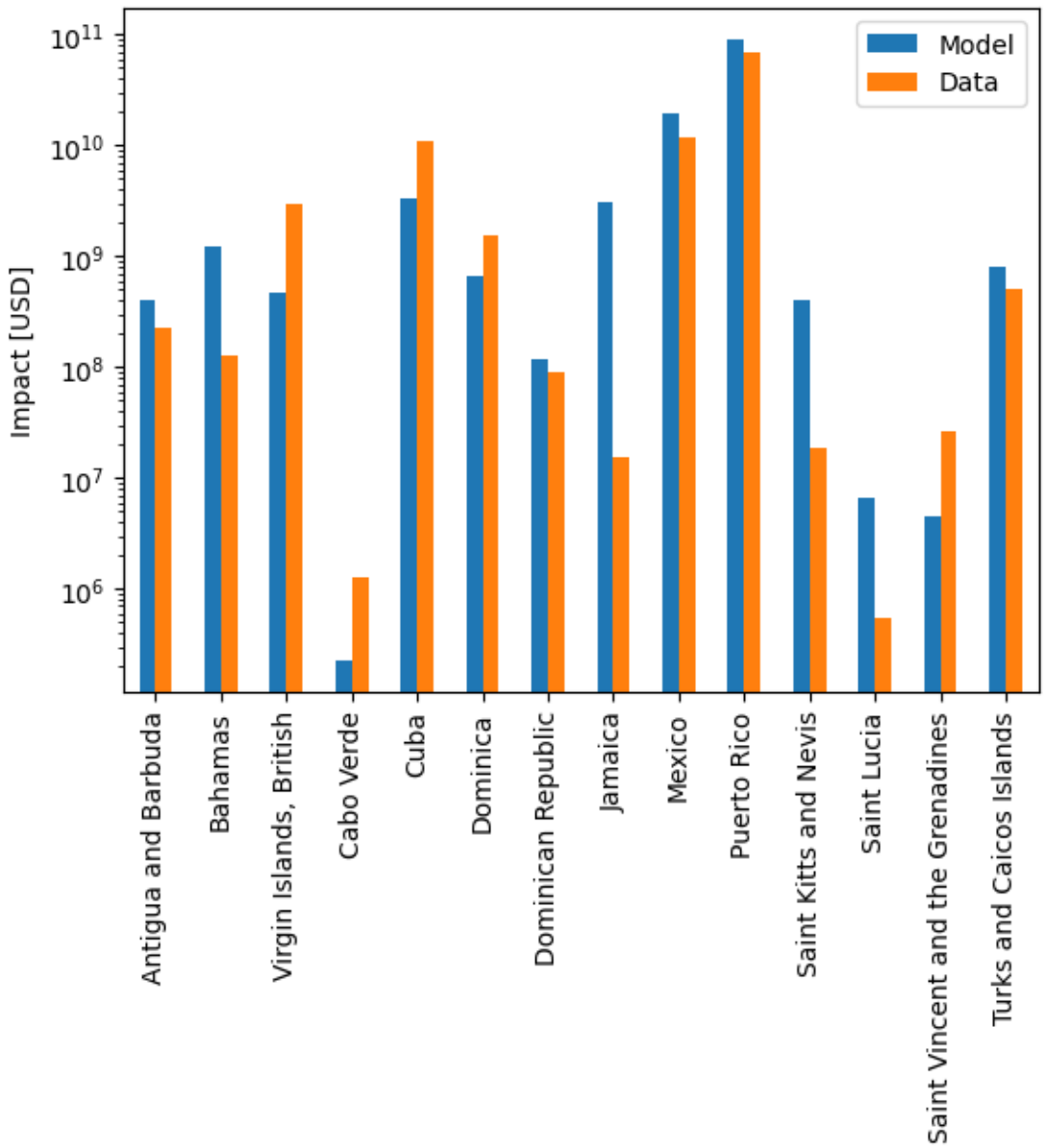
We can now compare the modelled and reported impact data on a country- or event-basis. The `OutputEvaluator` also has methods for that. In both of these, you can supply a transformation function with the `data_transf` argument. This transforms the data to be plotted right before plotting. Recall that we set the event IDs as index for the data frames. To better interpret the results, it is useful to transform them into event names again, which are the IBTrACS IDs. Likewise, we use the region IDs for region identification. It might be nicer to transform these into country names before plotting.

```
import climada.util.coordinates as u_coord


def country_code_to_name(code):
    return u_coord.country_to_iso(code, representation="name")


event_id_to_name = {
    hazard.event_id[idx]: hazard.event_name[idx] for idx in range(len(hazard.event_
↪id))
}
```

```
output_eval.plot_at_event(
    data_transf=lambda x: x.rename(index=event_id_to_name), logy=True
)
output_eval.plot_at_region(
    data_transf=lambda x: x.rename(index=country_code_to_name), logy=True
)
```

```
<Axes: ylabel='Impact [USD]'>
```

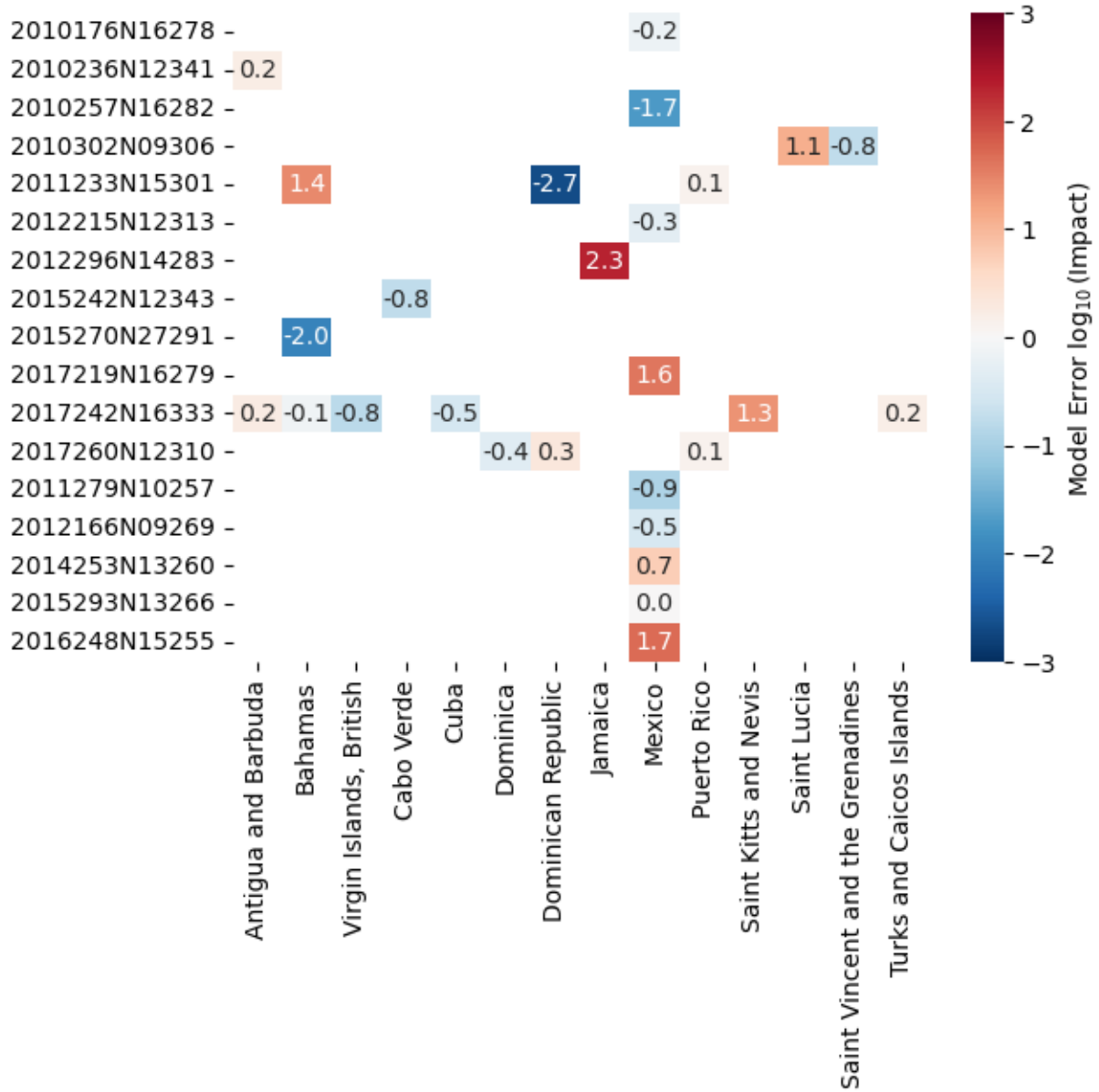

Model
Data
Impact [USD]
10^{11}
10^{10}
10^{9}
10^{8}
10^{7}
10^{6}
2010176N16278
2010236N12341
2010257N16282
2010302N09306
2011233N15301
2012215N12313
2012296N14283
2015242N12343
2015270N27291
2017219N16279
2017242N16333
2017260N12310
2011279N10257
2012166N09269
2014253N13260
2015293N13266
2016248N15255



Finally, we can do an event- *and* country-based comparison using a heatmap using `plot_event_region_heatmap()` Since the magnitude of the impact values may differ strongly, this method compare them on a logarithmic scale. It divides each modelled impact by the observed impact and takes the the decadic logarithm. The result will tell us how many orders of magnitude our model was off. Again, the considerations for "nicer" index and columns apply.

```
output_eval.plot_event_region_heatmap(
    data_transf=lambda x: x.rename(index=event_id_to_name, columns=country_code_to_
↪name)
)
```

```
<Axes: >
```



2.9.8 Handling Missing Data

NaN-valued input data has a special meaning in calibration: It implies that the impact calculated by the model for that data point should be ignored. Opposed to that, a value of zero indicates that the model should be calibrated towards an impact of exactly zero for this data point.

There might be instances where data is provided for a certain region or event, and the model produces no impact for them. This is always treated as an impact of zero during the calibration. Likewise, there might be instances where the model computes an impact for a region for which no impact data is available. In these cases, `missing_data_value` of `Input` is used as fill value for the data. According to the data value logic mentioned above, `missing_data_value=np.nan` (the default setting) will cause the modeled impact to be ignored in the calibration. Setting `missing_data_value=0`, on the other hand, will calibrate the model towards zero impact for all regions or events where no data is supplied.

Let's exemplify this with a subset of the data used for the last calibration. Irma is the cyclone with ID `2017242N16333`, and it hit most locations we were looking at before. It has the event ID `1454` in our setup. Let's just use this hazard and

the related data for now.

NOTE: We must pass a dataframe to the input, but selecting a single row or column will return a Series. We expand it into a dataframe again and make sure the new frame is oriented the correct way.

```
hazard_irma = all_tcs.select(event_names=["2017242N16333"])
data_irma = data.loc[1454, :].to_frame().T
data_irma
```

```
region_id          28           44            92   132           192  212  \
1454       211176356.8  1801876.321  3.000000e+09  NaN  1.099275e+10  NaN

region_id  214  388  484  630          659  662  670          796
1454       NaN  NaN  NaN  NaN  18484889.46  NaN  NaN  500000000.0
```

Let's first calibrate the impact function only on this event, including all data we have.

```
from climada.util.calibrate import (
    Input,
    BayesianOptimizer,
    OutputEvaluator,
    msle,
)
import matplotlib.pyplot as plt


def calibrate(hazard, data, **input_kwargs):
    """Calibrate using custom hazard and data"""
    # Define calibration input
    input = Input(
        hazard=hazard,
        exposure=exposure,
        data=data,
        impact_func_creator=impact_func_tc,
        cost_func=msle,
        impact_to_dataframe=lambda imp: imp.impact_at_reg(exposure.gdf["region_id"]),
        bounds=bounds,
        **input_kwargs,
    )

    # Create and run the optimizer
    with log_level("INFO", name_prefix="climada.util.calibrate"):
        opt = BayesianOptimizer(input)
        controller = BayesianOptimizerController.from_input(input)
        bayes_output = opt.run(controller=controller)

    # Evaluate output
    output_eval = OutputEvaluator(input, bayes_output)
    output_eval.impf_set.plot()

    plt.figure()  # New figure because seaborn.heatmap draws into active axes
    output_eval.plot_event_region_heatmap(
        data_transf=lambda x: x.rename(
            index=event_id_to_name, columns=country_code_to_name
```

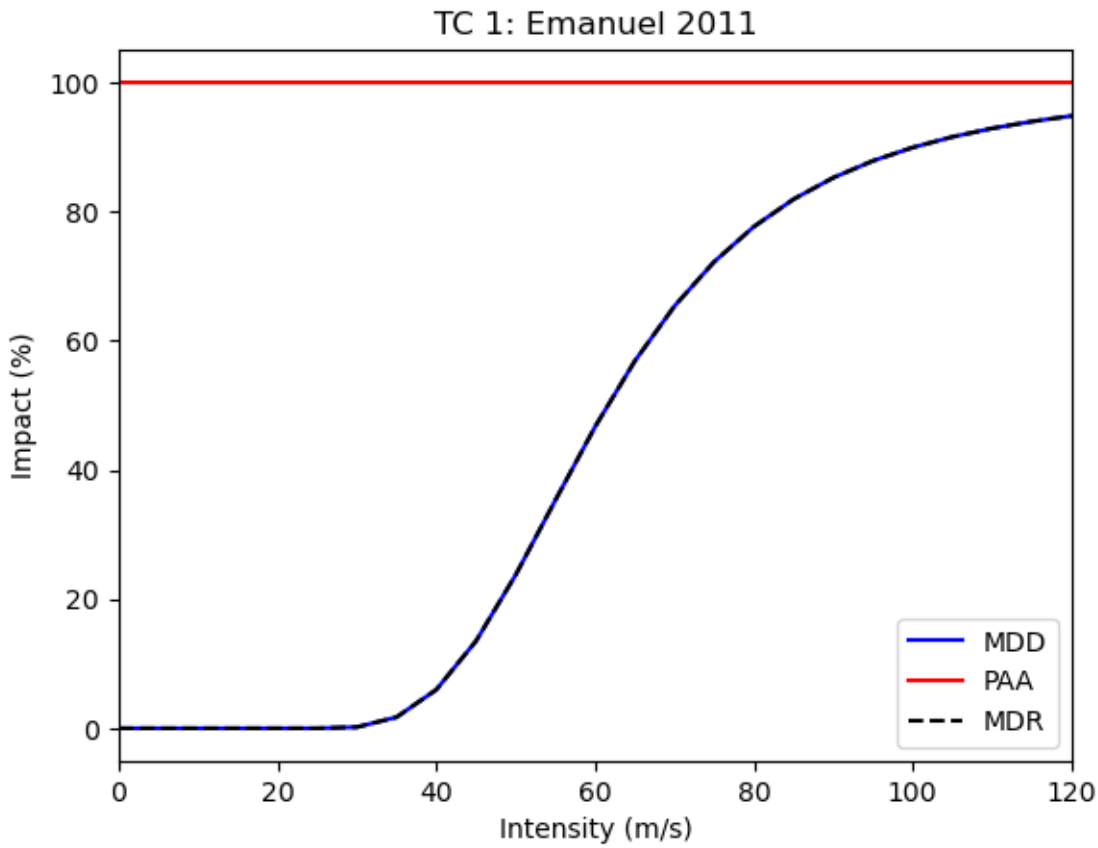
(continues on next page)

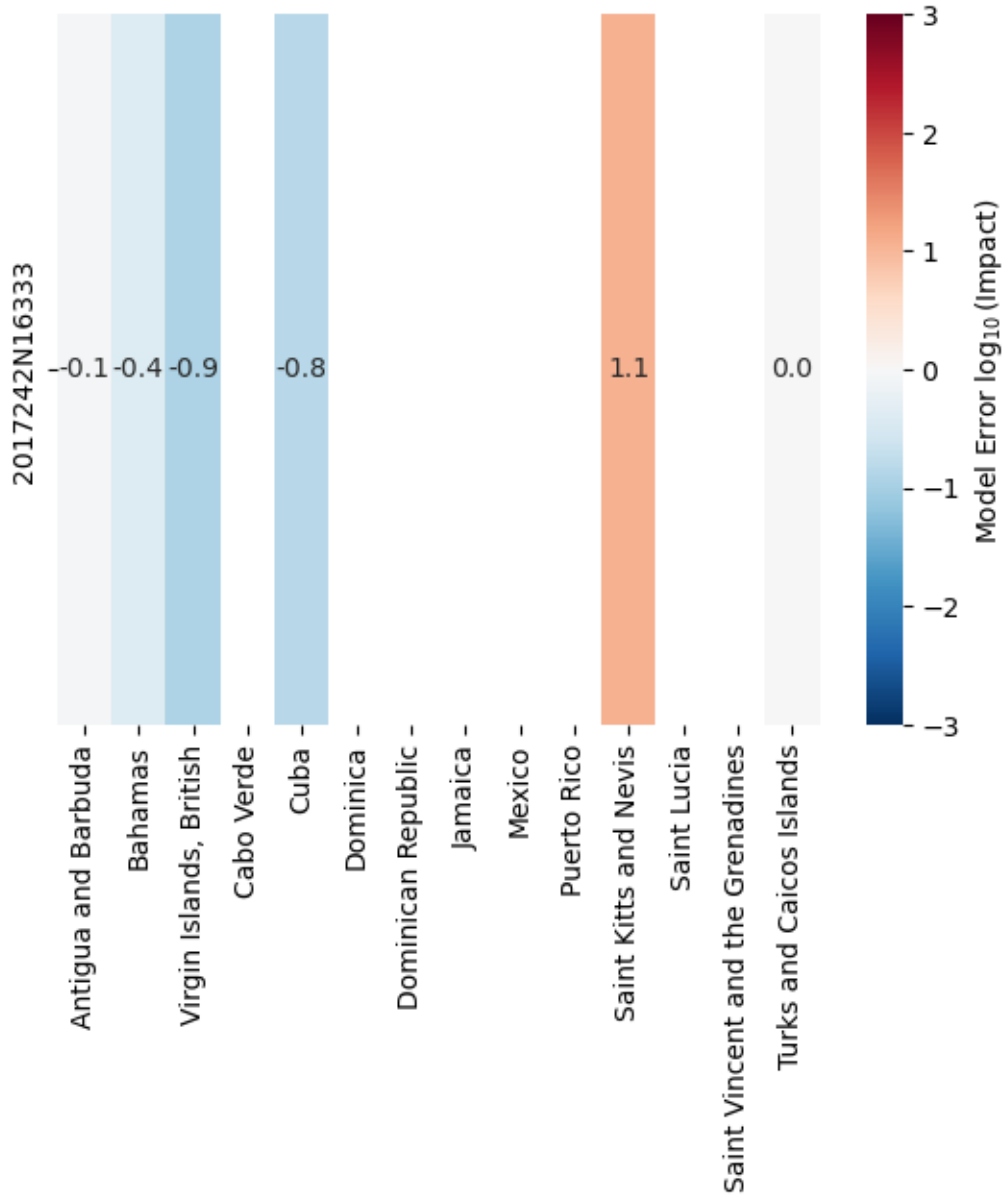
(continued from previous page)

```
        )
    )

    return bayes_output.params  # The optimal parameters
```

```
param_irma = calibrate(hazard_irma, data_irma)
```

```
2025-04-25 14:27:18,449 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 0
2025-04-25 14:27:20,708 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 1
2025-04-25 14:27:23,082 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 2
2025-04-25 14:27:25,550 - climada.util.calibrate.bayesian_optimizer - INFO - No␣
↪improvement. Stop optimization.
```





If we now remove some of the damage reports and repeat the calibration, the respective impact computed by the model will be ignored. For Saint Kitts and Nevis, and for Turks and the Caicos Islands, the impact is overestimated by the model. Removing these regions from the estimation should shift the estimated parameters accordingly, because by default, impacts for missing data points are ignored with `missing_data_value=np.nan`.

```
calibrate(hazard_irma, data_irma.drop(columns=[659, 796]))
```

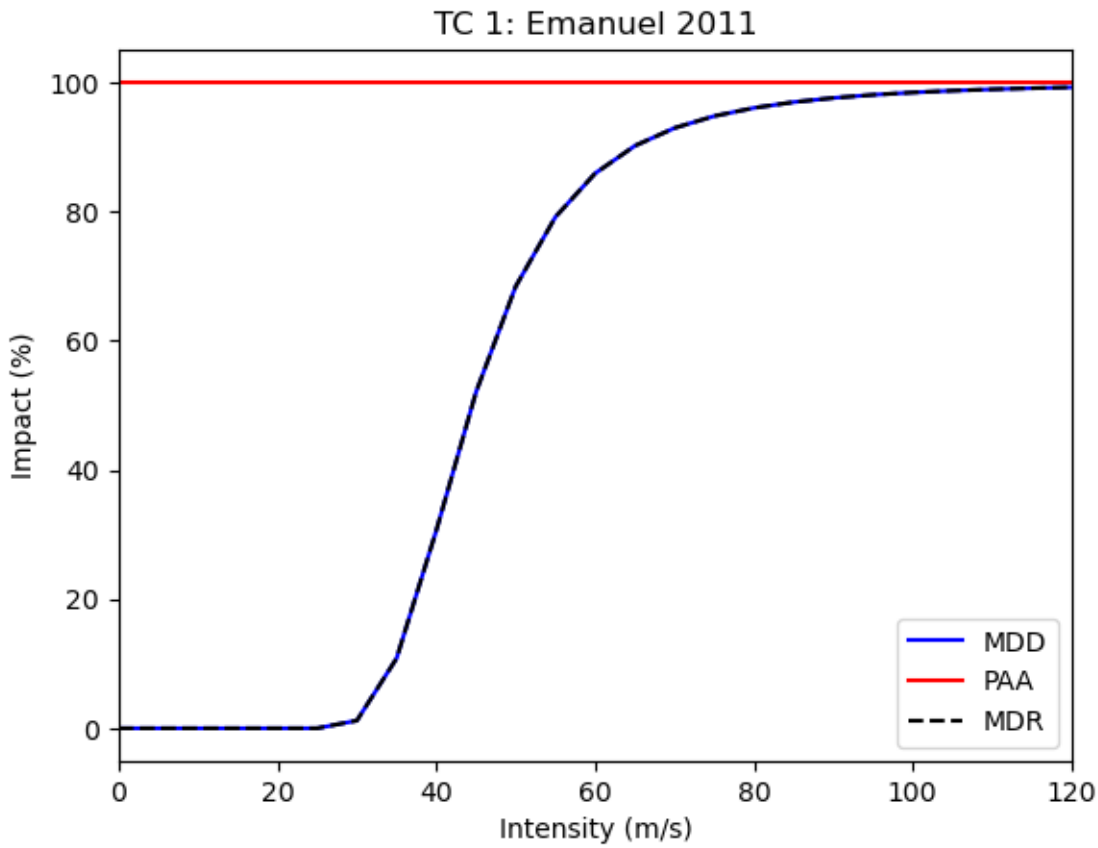
```
2025-04-25 14:27:38,008 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 0
2025-04-25 14:27:40,210 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 1
2025-04-25 14:27:42,394 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 2
2025-04-25 14:27:44,707 - climada.util.calibrate.bayesian_optimizer - INFO - No␣
```

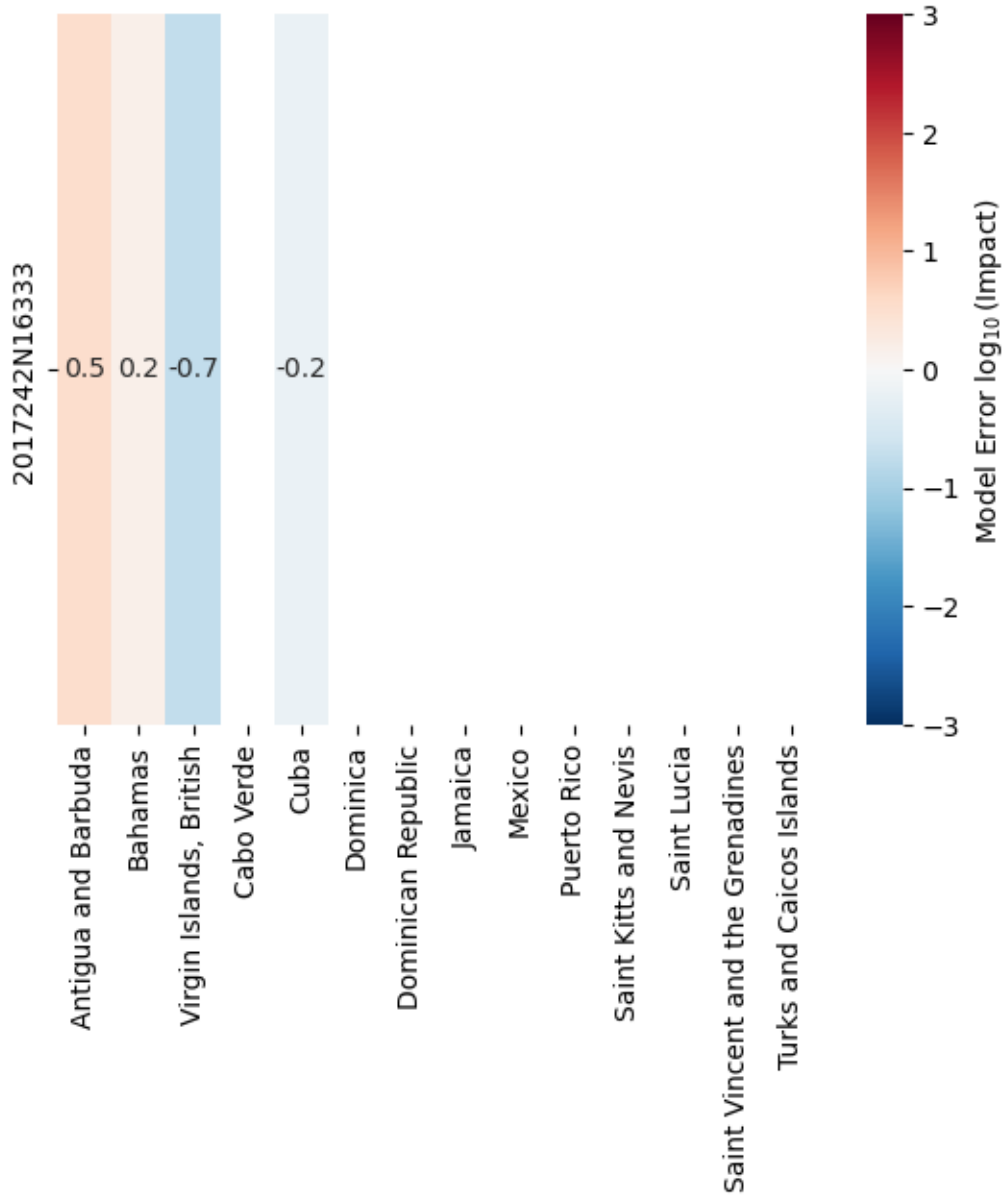
(continues on next page)

(continued from previous page)

```
↪improvement. Stop optimization.
```

```
{'scale': 1.0, 'v_half': 44.48812601602297}
```



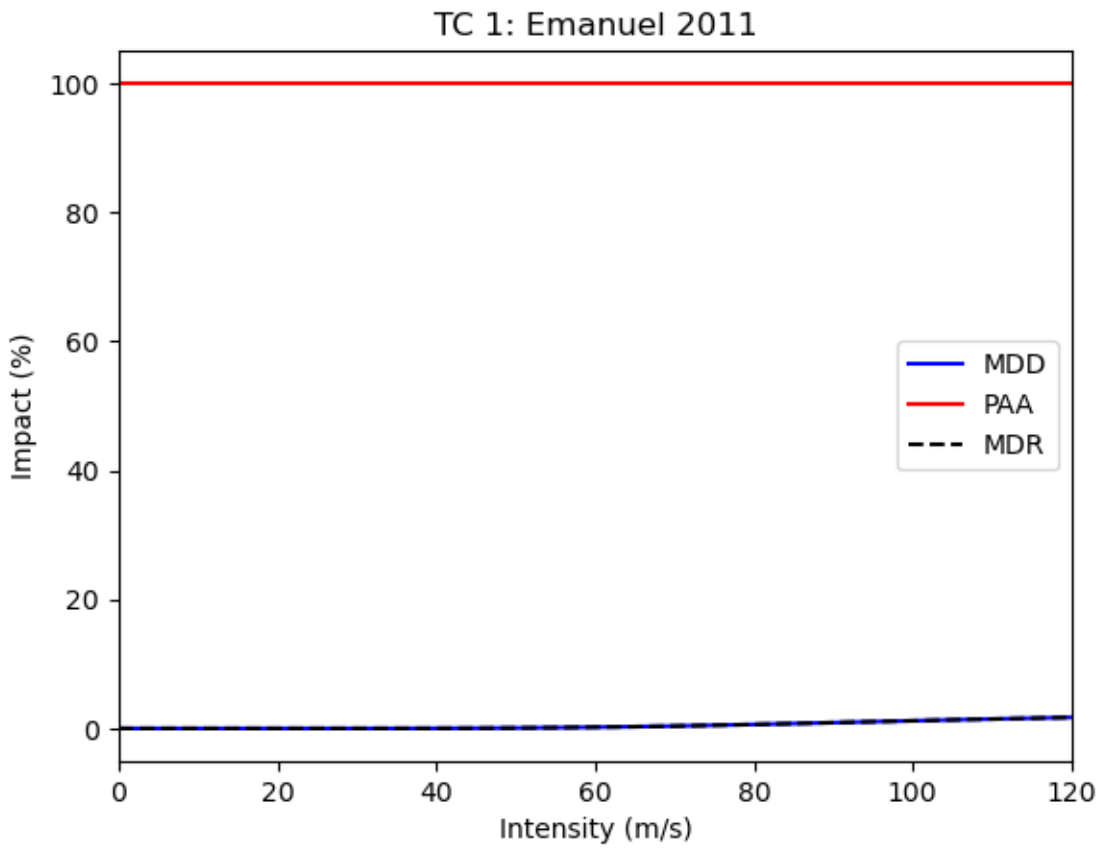


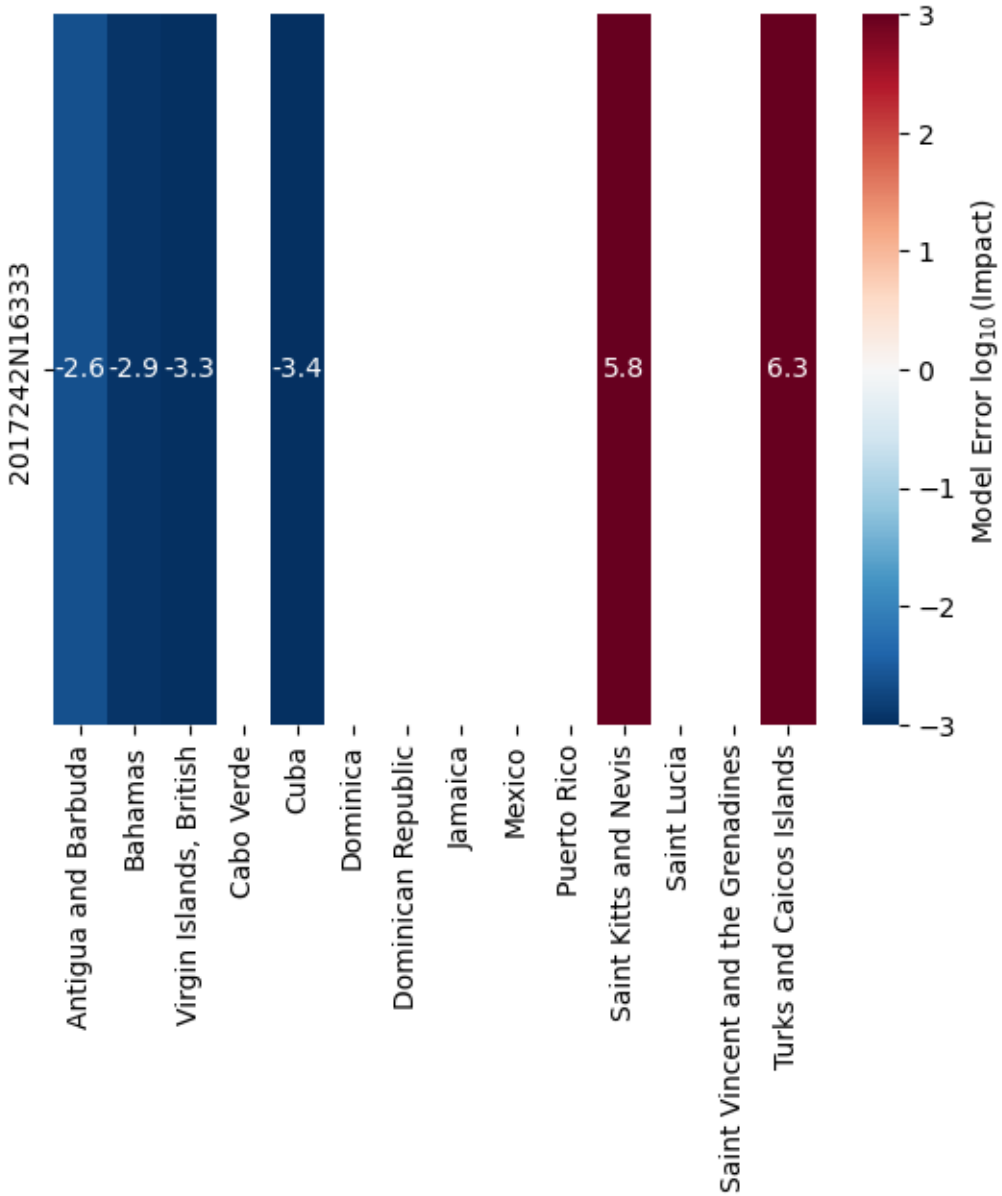
However, the calibration should change into the other direction once we require the modeled impact at missing data points to be zero:

```
calibrate(hazard_irma, data_irma.drop(columns=[659, 796]), missing_data_value=0)
```

```
2025-04-25 14:27:58,736 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 0
2025-04-25 14:28:01,264 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 1
2025-04-25 14:28:04,332 - climada.util.calibrate.bayesian_optimizer - INFO -␣
↪Optimization iteration: 2
2025-04-25 14:28:07,387 - climada.util.calibrate.bayesian_optimizer - INFO - No␣
↪improvement. Stop optimization.
```

```
{'scale': 0.029173288291594105, 'v_half': 110.11137319113446}
```





Actively requiring that the model calibrates towards zero impact in the two dropped regions means that it will typically strongly overestimate the impact there (because impact actually took place). This will "flatten" the vulnerability curve, causing strong underestimation in the other regions.

2.9.9 How to Continue

While the found impact function looks reasonable, we find that the model overestimates the impact severely for several events. This might be due to missing data, but it is also strongly related to the choice of impact function (shape) and the particular goal of the calibration task.

The most crucial information in the calibration task is the cost function. The RMSE measure is sensitive to the largest errors (and hence the largest impact). Therefore, using it in the calibration minimizes the overall error, but will incorrectly capture events with impact of lower orders of magnitude. Using a cost function based on the ratio between modelled and observed impact might increase the overall error but decrease the log-error for many events.

So we present some ideas on how to continue and/or improve the calibration:

1. Run the calibration again, but change the number of initial steps and/or iteration steps.
2. Use a different cost function, e.g., an error measure based on a ratio rather than a difference.
3. Also calibrate the `v_thresh` parameter. This requires adding constraints, because `v_thresh` < `v_half`.
4. Calibrate different impact functions for houses in Mexico and Puerto Rico within the same optimization task.
5. Employ the *`ScipyMinimizeOptimizer`* instead of the `BayesianOptimizer`.

2.9.10 Ensemble Calibration

The parameter space of a single calibration task gives an estimate of the uncertainty within that calibration. However, it only gives limited information on the uncertainty stemming from the model setup and impact data provided. To estimate the full model-chain uncertainty, we provide additional method for calibrating ensembles of impact functions based on subsets of the full impact data.

We define two types of ensembles and hence calibration tasks:

- The *average ensemble* (AE) collects impact functions that are calibrated on sampled subsets of the impact data.
- The *ensemble of tragedies* (EoT) collects impact functions that are calibrated on a single impact record each.

We provide two classes that calibrate these types of ensembles, namely *`AverageEnsembleOptimizer`* and *`TragedyEnsembleOptimizer`*. These classes define how the impact data is sampled for each individual impact function calibration task, and then defer this task to an underlying optimizer (which we showcased before). Both take *`Input`* object that could also be used in a single calibration optimizer.

Average Ensemble

For each calibration of the AE, we sample the impact data by drawing a fraction of all impact values. In this case, let's draw the default 80% with an ensemble size of 5, meaning that we compute 5 distinct impact functions. As underlying optimizer, we again choose the *`BayesianOptimizer`*. The *`BayesianOptimizerController`* is passed via the `optimizer_run_kwargs` argument of *`run()`*. Note that you must provide the *type* of the underlying optimizer when constructing an `EnsembleOptimizer`.

```
from climada.util.calibrate import (
    Input,
    AverageEnsembleOptimizer,
    BayesianOptimizer,
    BayesianOptimizerController,
    msle,
)
from climada.util import log_level

# Define calibration input
with log_level("WARNING", name_prefix="climada.util.calibrate"):
    input = Input(
        hazard=hazard,
        exposure=exposure,
        data=data,
        impact_func_creator=impact_func_tc,
        cost_func=msle,
        impact_to_dataframe=lambda imp: imp.impact_at_reg(exposure.gdf["region_id"]),
        bounds=bounds,
    )

    # Create and run the optimizer
```

(continues on next page)

(continued from previous page)

```
    opt_ae = AverageEnsembleOptimizer(input, optimizer_type=BayesianOptimizer)
    controller = BayesianOptimizerController.from_input(input)
    output_ae = opt_ae.run(controller=controller)
```

```
100%|██████████| 20/20 [03:01<00:00,  9.08s/it]
```

The output of an ensemble optimizer is different from that of a single optimizer. Instead of the parameter space for each optimization we only provide the best estimate of each iteration, along with some information on the events and regions the particular function was calibrated on. We also provide methods for plotting the resulting ensemble of impact functions.

```
output_ae.data
```

```
   Parameters                                                          Event  \
        scale     v_half                                             event_id
0    0.222773  37.142783  [1333, 1344, 1351, 1361, 3686, 1390, 3743, 142...
1    1.000000  51.623892  [1333, 1339, 1344, 1351, 1361, 3686, 1377, 374...
2    1.000000  62.106583  [1339, 1351, 1361, 3686, 3691, 1390, 3743, 142...
3    1.000000  61.436076  [1339, 1351, 1361, 3691, 1421, 1426, 3777, 379...
4    0.060143  31.095014  [1333, 1344, 1361, 3686, 3691, 1390, 3743, 379...
5    1.000000  61.224231  [1333, 1339, 1351, 3691, 1390, 3743, 1421, 377...
6    0.819839  47.117318  [1333, 1339, 1351, 1361, 3686, 1377, 3743, 142...
7    1.000000  54.945455  [1333, 1344, 1351, 1361, 3686, 3691, 1377, 139...
8    0.825518  47.235135  [1339, 1344, 1351, 1361, 3686, 1377, 1390, 374...
9    1.000000  53.537666  [1339, 1344, 1351, 1361, 3686, 3691, 1390, 374...
10   1.000000  46.770608  [1344, 1351, 1361, 3691, 1421, 3777, 3795, 145...
11   0.564386  42.199320  [1344, 1351, 1361, 1377, 1421, 1426, 3777, 145...
12   1.000000  54.550104  [1333, 1351, 1361, 3686, 3691, 1377, 1426, 379...
13   0.094194  30.650604  [1333, 1361, 3686, 1377, 1390, 3743, 1421, 379...
14   0.517233  49.958477  [1333, 1351, 1361, 3691, 1377, 1390, 3743, 377...
15   0.289714  30.852674  [1339, 1344, 1351, 1361, 3691, 1426, 3777, 379...
16   0.749633  53.316453  [1333, 1339, 1351, 1361, 3691, 1390, 3743, 142...
17   1.000000  54.641313  [1333, 1344, 1351, 1361, 3686, 3691, 1377, 374...
18   0.812884  51.297538  [1333, 1339, 1344, 1351, 1361, 1377, 1390, 374...
19   1.000000  54.085105  [1333, 1339, 1344, 1351, 1361, 3686, 3691, 137...

                                                        \
                                                region_id
0    [44, 92, 132, 214, 388, 484, 630, 659, 662, 796]
1     [28, 44, 92, 132, 192, 212, 214, 484, 630, 662]
2   [28, 44, 92, 132, 212, 388, 484, 630, 662, 670...
3     [28, 44, 92, 132, 192, 484, 630, 659, 662, 796]
4         [28, 44, 212, 214, 388, 484, 630, 659, 796]
5   [28, 132, 192, 212, 214, 388, 484, 630, 659, 6...
6   [28, 44, 92, 132, 214, 484, 630, 659, 662, 670...
7         [44, 92, 132, 192, 388, 484, 630, 670, 796]
8     [28, 44, 92, 212, 214, 388, 484, 630, 662, 670]
9   [28, 44, 92, 192, 212, 388, 484, 630, 659, 662...
10    [28, 44, 92, 132, 192, 212, 214, 484, 630, 662]
11  [28, 44, 132, 192, 212, 214, 484, 630, 659, 66...
12         [28, 44, 92, 192, 214, 484, 630, 662, 796]
```

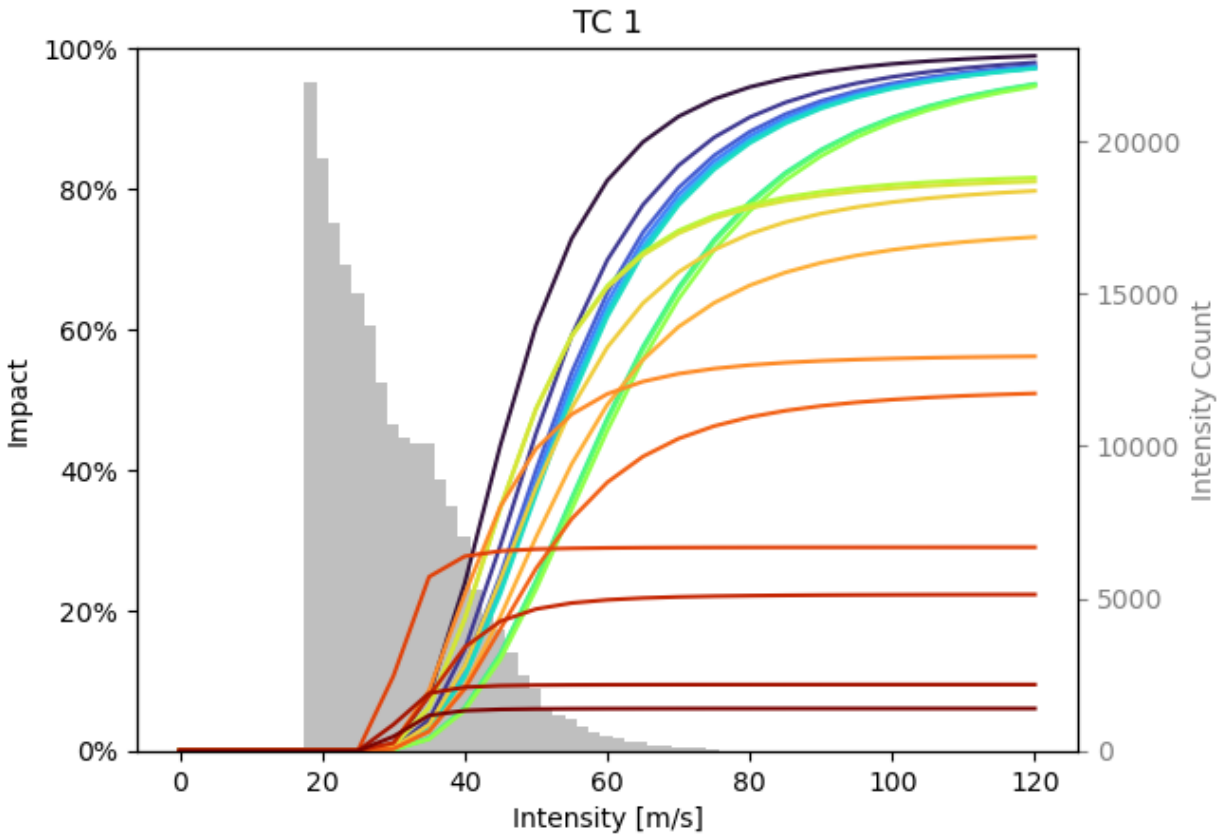
(continues on next page)

(continued from previous page)

```
13  [44, 92, 132, 192, 212, 214, 388, 484, 630, 65...
14         [28, 44, 92, 192, 214, 388, 484, 630, 662]
15  [28, 44, 92, 192, 212, 214, 484, 630, 659, 670...
16  [28, 44, 132, 192, 212, 214, 388, 484, 630, 66...
17  [28, 44, 92, 192, 214, 484, 630, 659, 662, 670...
18  [28, 44, 92, 192, 212, 214, 388, 484, 630, 662...
19  [28, 44, 92, 132, 192, 212, 388, 484, 630, 659...


                                            event_name
0   [2010176N16278, 2010257N16282, 2010302N09306, ...
1   [2010176N16278, 2010236N12341, 2010257N16282, ...
2   [2010236N12341, 2010302N09306, 2011233N15301, ...
3   [2010236N12341, 2010302N09306, 2011233N15301, ...
4   [2010176N16278, 2010257N16282, 2011233N15301, ...
5   [2010176N16278, 2010236N12341, 2010302N09306, ...
6   [2010176N16278, 2010236N12341, 2010302N09306, ...
7   [2010176N16278, 2010257N16282, 2010302N09306, ...
8   [2010236N12341, 2010257N16282, 2010302N09306, ...
9   [2010236N12341, 2010257N16282, 2010302N09306, ...
10  [2010257N16282, 2010302N09306, 2011233N15301, ...
11  [2010257N16282, 2010302N09306, 2011233N15301, ...
12  [2010176N16278, 2010302N09306, 2011233N15301, ...
13  [2010176N16278, 2011233N15301, 2011279N10257, ...
14  [2010176N16278, 2010302N09306, 2011233N15301, ...
15  [2010236N12341, 2010257N16282, 2010302N09306, ...
16  [2010176N16278, 2010236N12341, 2010302N09306, ...
17  [2010176N16278, 2010257N16282, 2010302N09306, ...
18  [2010176N16278, 2010236N12341, 2010257N16282, ...
19  [2010176N16278, 2010236N12341, 2010257N16282, ...
```

```
output_ae.plot_shiny(
    impact_func_creator=impact_func_tc,
    haz_type="TC",
    impf_id=1,
    inp=input,
    legend=False,
)
```

```
<Axes: title={'center': 'TC 1'}, xlabel='Intensity [m/s]', ylabel='Impact'>
```



As you can see, although we only resampled the calibration data for each calibration, the functions differ quite a lot. We can investigate exactly which impact records cause this through the ensemble of tragedies. It can be calibrated the same way. By default, it computes one function for every data point in `data`.

```
from climada.util.calibrate import TragedyEnsembleOptimizer

# Define calibration input
with log_level("WARNING", name_prefix="climada.util.calibrate"):
    input = Input(
        hazard=hazard,
        exposure=exposure,
        data=data,
        impact_func_creator=impact_func_tc,
        cost_func=msle,
        impact_to_dataframe=lambda imp: imp.impact_at_reg(exposure.gdf["region_id"]),
        bounds=bounds,
    )

    # Create and run the optimizer
    opt_eot = TragedyEnsembleOptimizer(input, optimizer_type=BayesianOptimizer)
    controller = BayesianOptimizerController.from_input(input)
    output_eot = opt_eot.run(controller=controller)
```

```
Data point [ 1.  25.8] is not unique. 1 duplicates registered. Continuing ...
Data point [ 1.  25.8] is not unique. 2 duplicates registered. Continuing ...
Data point [ 1.  25.8] is not unique. 3 duplicates registered. Continuing ...
```

```
Data point [ 1.  25.8] is not unique. 1 duplicates registered. Continuing ...
Data point [ 1.  25.8] is not unique. 2 duplicates registered. Continuing ...
Data point [ 1.  25.8] is not unique. 3 duplicates registered. Continuing ...
```

```
100%|██████████| 27/27 [03:25<00:00,  7.61s/it]
```

```
output_eot.data
```

```
   Parameters                Event
        scale      v_half event_id region_id       event_name
0    0.623918   43.558467   [1333]     [484]  [2010176N16278]
1    0.446720   49.853994   [1339]      [28]  [2010236N12341]
2    0.870346   30.882206   [1344]     [484]  [2010257N16282]
3    0.332453   68.510883   [1351]     [662]  [2010302N09306]
4    0.659666   38.132968   [1351]     [670]  [2010302N09306]
5    0.501478   92.838101   [1361]      [44]  [2011233N15301]
6    1.000000   25.800000   [1361]     [214]  [2011233N15301]
7    0.577457   50.142499   [1361]     [630]  [2011233N15301]
8    0.285608   31.604609   [3686]     [484]  [2011279N10257]
9    0.257775   30.957414   [3691]     [484]  [2012166N09269]
10   0.409814   40.494643   [1377]     [484]  [2012215N12313]
11   0.455964  148.659015   [1390]     [388]  [2012296N14283]
12   0.190400   50.537864   [3743]     [484]  [2014253N13260]
13   0.665652   37.430581   [1421]     [132]  [2015242N12343]
14   1.000000   25.800000   [1426]      [44]  [2015270N27291]
15   0.563058   46.037049   [3777]     [484]  [2015293N13266]
16   0.522021  112.298359   [3795]     [484]  [2016248N15255]
17   1.000000  117.135848   [1450]     [484]  [2017219N16279]
18   0.473914   50.544738   [1454]      [28]  [2017242N16333]
19   0.193221   30.347552   [1454]      [44]  [2017242N16333]
20   1.000000   32.463875   [1454]      [92]  [2017242N16333]
21   0.482094   30.424033   [1454]     [192]  [2017242N16333]
22   0.142128   69.089890   [1454]     [659]  [2017242N16333]
23   1.000000   62.107956   [1454]     [796]  [2017242N16333]
24   1.000000   26.131008   [1458]     [212]  [2017260N12310]
25   0.399726   51.035757   [1458]     [214]  [2017260N12310]
26   0.554834   37.334453   [1458]     [630]  [2017260N12310]
```

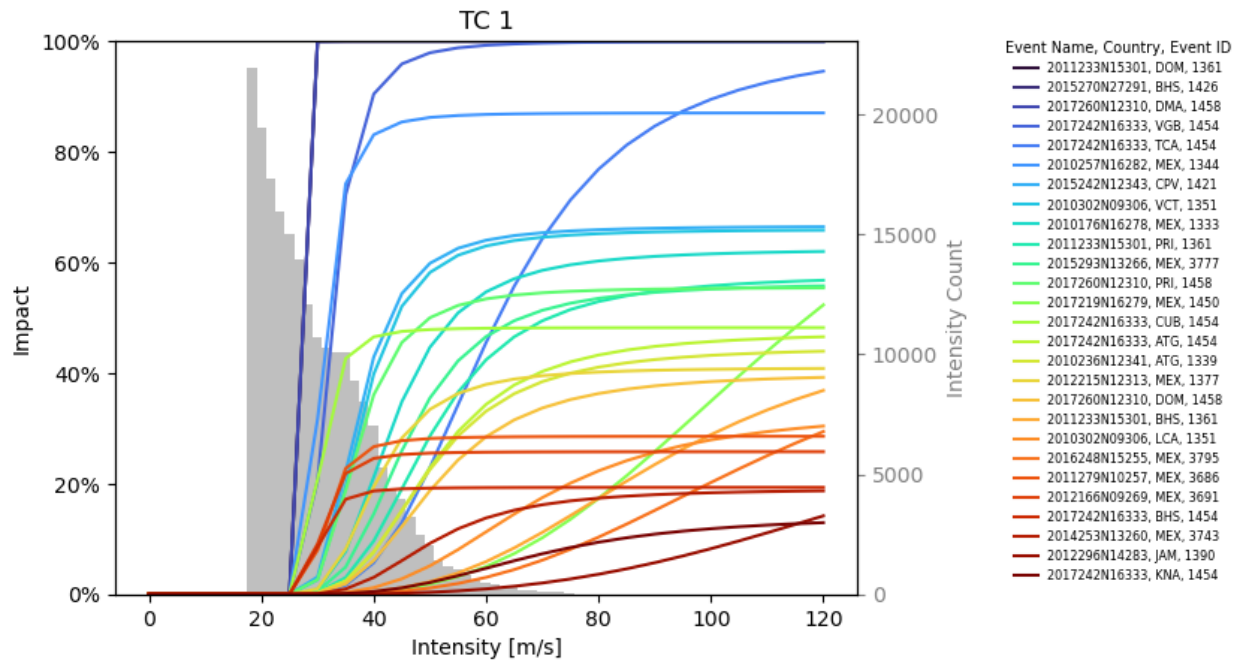
```
output_eot.plot_shiny(
    impact_func_creator=impact_func_tc,
    haz_type="TC",
    impf_id=1,
    inp=input,
    legend=True,
```

(continues on next page)

(continued from previous page)

```
)
```

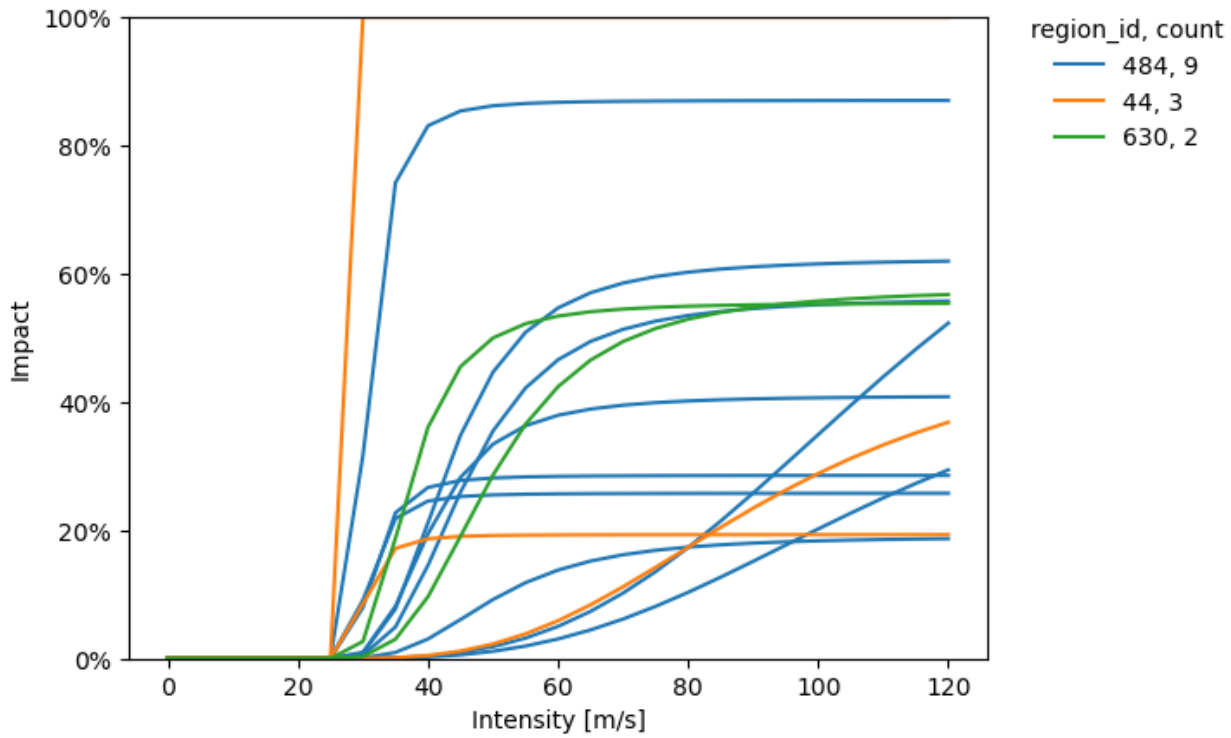
```
<Axes: title={'center': 'TC 1'}, xlabel='Intensity [m/s]', ylabel='Impact'>
```



The `plot_category` function helps to differentiate between impact functions by plotting functions of one category with the same color. Any data in the `Event` column of the `data` attribute in the output data may be used.

```
from climada.util.coordinates import country_to_iso


def to_iso(country):
    return country_to_iso(country, "numeric")


output_eot.plot_category(
    impact_func_creator=impact_func_tc,
    haz_type="TC",
    impf_id=1,
    category="region_id",
    category_colors={to_iso("MEX"): "C0", to_iso("BHS"): "C1", to_iso("PRI"): "C2"},
)
```

```
<Axes: xlabel='Intensity [m/s]', ylabel='Impact'>
```



2.10 Google Earth Engine (GEE) and Image Analysis

This tutorial explains how to use the module ***climada.util.earth_engine***. It queries data from the Google Earth Engine Python API (https://earthengine.google.com/). A few basic methods of image processing will also be presented using algorythms from Scikit-image (https://scikit-image.org/). A lot of complementary information can be found on this page https://developers.google.com/earth-engine/ (concerns mostly the GEE Java API, but concept and methods are well detailed). GEE is a multi-petabyte catalog of satellite imagery and geospatial datasets. The data are also available on the website of providers, GEE is just more user-friendly as all datasets are available through the same platform.

2.10.1 Connect to Google Earth Engine API

To access the data, you have to create an account on https://signup.earthengine.google.com/#!/, this step might take some time. Then, install and connect your Python to the API using the terminal. Be sure that climada_env is activated.

In Terminal or Anaconda prompt

$ source activate climada_env

$ conda install -c conda-forge earthengine-api

Then, when the installation is finished, type

$ earthengine authenticate

This will open a web page where you have to enter your account information and a code is provided. Paste it in the terminal.

Then, check in Python if it has worked with the lines below. Import also webbrowser for further steps.

```
import webbrowser
```

(continues on next page)

(continued from previous page)

```
import ee

ee.Initialize()
image = ee.Image("srtm90_v4")
print(image.getInfo())
```

```
{'type': 'Image', 'bands': [{'id': 'elevation', 'data_type': {'type': 'PixelType',
↪'precision': 'int', 'min': -32768, 'max': 32767}, 'dimensions': [432000, 144000],
↪'crs': 'EPSG:4326', 'crs_transform': [0.000833333333333, 0, -180, 0, -0.
↪000833333333333, 60]}], 'version': 1494271934303000.0, 'id': 'srtm90_v4',
↪'properties': {'system:time_start': 950227200000, 'system:time_end': 951177600000,
↪'system:asset_size': 18827626666}}
```

2.10.2 Obtain images

The module ***climada.util.earth_engine*** enables to select images from some collections of GEE and download them as Geotiff data.

In GEE, you can either access directly one **image** or a **collection**. All products available are detailed on this page https://developers.google.com/earth-engine/datasets/.

```
# Access a specific image
image = ee.Image("LANDSAT/LC08/C01/T1_TOA/LC08_044034_20140318")
# Landsat 8 image, with Top of Atmosphere processing, on 2014/03/18

# Access a collection
collection = "LANDSAT/LE07/C01/T1"  # Landsat 7 raw images collection
```

If you have a collection, specification of the time range and area of interest. Then, use methods of the series **obtain_image_type(collection,time_range,area)** depending the type of product needed.

Time range

It depends on the image acquisition period of the targeted satellite and type of images desired (without clouds, from a specific period…)

Area

GEE needs a special format for defining an area of interest. It has to be a GeoJSON Polygon and the coordinates should be first defined in a list and then converted using ee.Geometry. It is possible to use data obtained via Exposure layer. Some examples are given below.

```
# Landsat_composite in Dresden area
area_dresden = list(
    [(13.6, 50.96), (13.9, 50.96), (13.9, 51.12), (13.6, 51.12), (13.6, 50.96)]
)
area_dresden = ee.Geometry.Polygon(area_dresden)
time_range_dresden = ["2002-07-28", "2002-08-05"]

collection_dresden = "LANDSAT/LE07/C01/T1"
print(type(area_dresden))

# Population density in Switzerland
```

(continues on next page)

(continued from previous page)

```
list_swiss = list(
    [(6.72, 47.88), (6.72, 46.55), (9.72, 46.55), (9.72, 47.88), (6.72, 47.88)]
)
area_swiss = ee.Geometry.Polygon(list_swiss)
time_range_swiss = ["2002-01-01", "2005-12-30"]

collection_swiss = ee.ImageCollection("CIESIN/GPWv4/population-density")
print(type(collection_swiss))

# Sentinel 2 cloud-free image in Zürich
collection_zurich = "COPERNICUS/S2"
list_zurich = list(
    [(8.53, 47.355), (8.55, 47.355), (8.55, 47.376), (8.53, 47.376), (8.53, 47.355)]
)
area_zurich = ee.Geometry.Polygon(list_swiss)
time_range_zurich = ["2018-05-01", "2018-07-30"]


# Landcover in Europe with CORINE dataset
dataset_landcover = ee.Image("COPERNICUS/CORINE/V18_5_1/100m/2012")
landCover_layer = dataset_landcover.select("landcover")
print(type(landCover_layer))
```

```
<class 'ee.geometry.Geometry'>
<class 'ee.imagecollection.ImageCollection'>
<class 'ee.image.Image'>
```

```
# Methods from climada.util.earth_engine module
def obtain_image_landsat_composite(collection, time_range, area):
    """Selection of Landsat cloud-free composites in the Earth Engine library
    See also: https://developers.google.com/earth-engine/landsat

    Parameters:
        collection (): name of the collection
        time_range (['YYYY-MT-DY','YYYY-MT-DY']): must be inside the available data
        area (ee.geometry.Geometry): area of interest

    Returns:
        image_composite (ee.image.Image)
    """
    collection = ee.ImageCollection(collection)

    ## Filter by time range and location
    collection_time = collection.filterDate(time_range[0], time_range[1])
    image_area = collection_time.filterBounds(area)
    image_composite = ee.Algorithms.Landsat.simpleComposite(image_area, 75, 3)
    return image_composite


def obtain_image_median(collection, time_range, area):
    """Selection of median from a collection of images in the Earth Engine library
```

(continues on next page)

(continued from previous page)

```
    See also: https://developers.google.com/earth-engine/reducers_image_collection

    Parameters:
        collection (): name of the collection
        time_range (['YYYY-MT-DY','YYYY-MT-DY']): must be inside the available data
        area (ee.geometry.Geometry): area of interest

    Returns:
        image_median (ee.image.Image)
    """
    collection = ee.ImageCollection(collection)

    ## Filter by time range and location
    collection_time = collection.filterDate(time_range[0], time_range[1])
    image_area = collection_time.filterBounds(area)
    image_median = image_area.median()
    return image_median


def obtain_image_sentinel(collection, time_range, area):
    """Selection of median, cloud-free image from a collection of images in the␣
↪Sentinel 2 dataset
    See also: https://developers.google.com/earth-engine/datasets/catalog/COPERNICUS_
↪S2

    Parameters:
        collection (): name of the collection
        time_range (['YYYY-MT-DY','YYYY-MT-DY']): must be inside the available data
        area (ee.geometry.Geometry): area of interest

    Returns:
        sentinel_median (ee.image.Image)
    """

    # First, method to remove cloud from the image
    def maskclouds(image):
        band_qa = image.select("QA60")
        cloud_mask = ee.Number(2).pow(10).int()
        cirrus_mask = ee.Number(2).pow(11).int()
        mask = band_qa.bitwiseAnd(cloud_mask).eq(0) and (
            band_qa.bitwiseAnd(cirrus_mask).eq(0)
        )
        return image.updateMask(mask).divide(10000)

    sentinel_filtered = (
        ee.ImageCollection(collection)
        .filterBounds(area)
        .filterDate(time_range[0], time_range[1])
        .filter(ee.Filter.lt("CLOUDY_PIXEL_PERCENTAGE", 20))
        .map(maskclouds)
    )
```

(continues on next page)

(continued from previous page)

```
    sentinel_median = sentinel_filtered.median()
    return sentinel_median
```

```
# Application to examples
composite_dresden = obtain_image_landsat_composite(
    collection_dresden, time_range_dresden, area_dresden
)
median_swiss = obtain_image_median(collection_swiss, time_range_swiss, area_swiss)
zurich_median = obtain_image_sentinel(collection_zurich, time_range_zurich, area_
↪zurich)

# Selection of specific bands from an image
zurich_band = zurich_median.select(["B4", "B3", "B2"])


print(composite_dresden.getInfo())
print(type(median_swiss))
print(type(zurich_band))
```

```
{'type': 'Image', 'bands': [{'id': 'B1', 'data_type': {'type': 'PixelType', 'precision
↪': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0,␣
↪1, 0]}, {'id': 'B2', 'data_type': {'type': 'PixelType', 'precision': 'int', 'min':␣
↪0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'B3
↪', 'data_type': {'type': 'PixelType', 'precision': 'int', 'min': 0, 'max': 255},
↪'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'B4', 'data_type':
↪{'type': 'PixelType', 'precision': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326',
↪ 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'B5', 'data_type': {'type': 'PixelType
↪', 'precision': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform':␣
↪[1, 0, 0, 0, 1, 0]}, {'id': 'B6_VCID_1', 'data_type': {'type': 'PixelType',
↪'precision': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1,␣
↪0, 0, 0, 1, 0]}, {'id': 'B6_VCID_2', 'data_type': {'type': 'PixelType', 'precision
↪': 'int', 'min': 0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0,␣
↪1, 0]}, {'id': 'B7', 'data_type': {'type': 'PixelType', 'precision': 'int', 'min':␣
↪0, 'max': 255}, 'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}, {'id': 'B8
↪', 'data_type': {'type': 'PixelType', 'precision': 'int', 'min': 0, 'max': 255},
↪'crs': 'EPSG:4326', 'crs_transform': [1, 0, 0, 0, 1, 0]}]}
<class 'ee.image.Image'>
<class 'ee.image.Image'>
```

2.10.3 Download images

To visualize and work on images, it is easier to download them (in Geotiff), using the **get_url(name, image, scale, region)** method. The image will be downloaded regarding a region and a scale. 'region' is obtained from the area, but the format has to be adjusted using **get_region(geom)** method.

```
def get_region(geom):
    """Get the region of a given geometry, needed for exporting tasks.

    Parameters:
        geom (ee.Geometry, ee.Feature, ee.Image): region of interest
```

(continues on next page)

(continued from previous page)

```
    Returns:
        region (list)
    """
    if isinstance(geom, ee.Geometry):
        region = geom.getInfo()["coordinates"]
    elif isinstance(geom, (ee.Feature, ee.Image)):
        region = geom.geometry().getInfo()["coordinates"]
    return region


region_dresden = get_region(area_dresden)
region_swiss = get_region(area_swiss)
region_zurich = get_region(area_zurich)
```

```
# If you want to apply this function to a list of regions:
region_list = [get_region(geom) for geom in [area_dresden, area_zurich]]
```

```
def get_url(name, image, scale, region):
    """It will open and download automatically a zip folder containing Geotiff data␣
↪of 'image'.
    If additional parameters are needed, see also:
    https://github.com/google/earthengine-api/blob/master/python/ee/image.py

    Parameters:
        name (str): name of the created folder
        image (ee.image.Image): image to export
        scale (int): resolution of export in meters (e.g: 30 for Landsat)
        region (list): region of interest

    Returns:
        path (str)
    """
    path = image.getDownloadURL({"name": (name), "scale": scale, "region": (region)})

    webbrowser.open_new_tab(path)
    return path


url_swiss = get_url("swiss_pop", median_swiss, 900, region_swiss)
url_dresden = get_url("dresden", composite_dresden, 30, region_dresden)
url_landcover = get_url("landcover_swiss", landCover_layer, 100, region_swiss)

# For the example of Zürich, due to size, it doesn't work on Jupyter Notebook but it␣
↪works on Python
# url_zurich = get_url('sentinel', zurich_band, 10, region_zurich)

print(url_swiss)
print(url_dresden)
print(url_landcover)
```

```
https://earthengine.googleapis.com/api/download?
↪docid=6b6d96f567d6a055188c8c17dd24bcb8&token=00a796601efe425c821777a284bff361
https://earthengine.googleapis.com/api/download?
↪docid=15182f82ba65ce24f62305e4465ac21c&token=5da59a20bb84d79bcf7ce958855fe848
https://earthengine.googleapis.com/api/download?
↪docid=07c14e22d96a33fc72a7ba16c2178a6a&token=0cfa0cd6537257e96600d10647375ff4
```

2.10.4 Image Visualization and Processing

In this section, basics methods of image processing will be presented as well as tools to visualize the image. The images downloaded before are used as examples but these methods works with all tif data. Scikit-image (https://scikit-image.org/) needs to be imported.

First, bands can be combined, for example to obtain an RGB image from the Red, Blue and Green bands. It is done with gdal_merge.py (see: https://gdal.org/programs/gdal_merge.html). It is better if bands are named just as B1, B2, B3 … in the folder containing the image data.

If you don't have any bands that you want to combine, you don't have to execute the following codes for this tutorial.

In Terminal or Anaconda prompt (be sure that climada_env is activated):

$ cd '/your/path/to/image_downloaded_folder'

$ gdal_merge.py -separate -co PHOTOMETRIC=RGB -o merged.tif B_red.tif B_blue.tif B_green.tif

The RGB image will be merged.tif

```
import numpy as np
from skimage import data
import matplotlib.pyplot as plt
from skimage.color import rgb2gray

from skimage.io import imread
from skimage import exposure
from skimage.filters import try_all_threshold
from skimage.filters import threshold_otsu, threshold_local
from skimage import measure
from skimage import feature
```

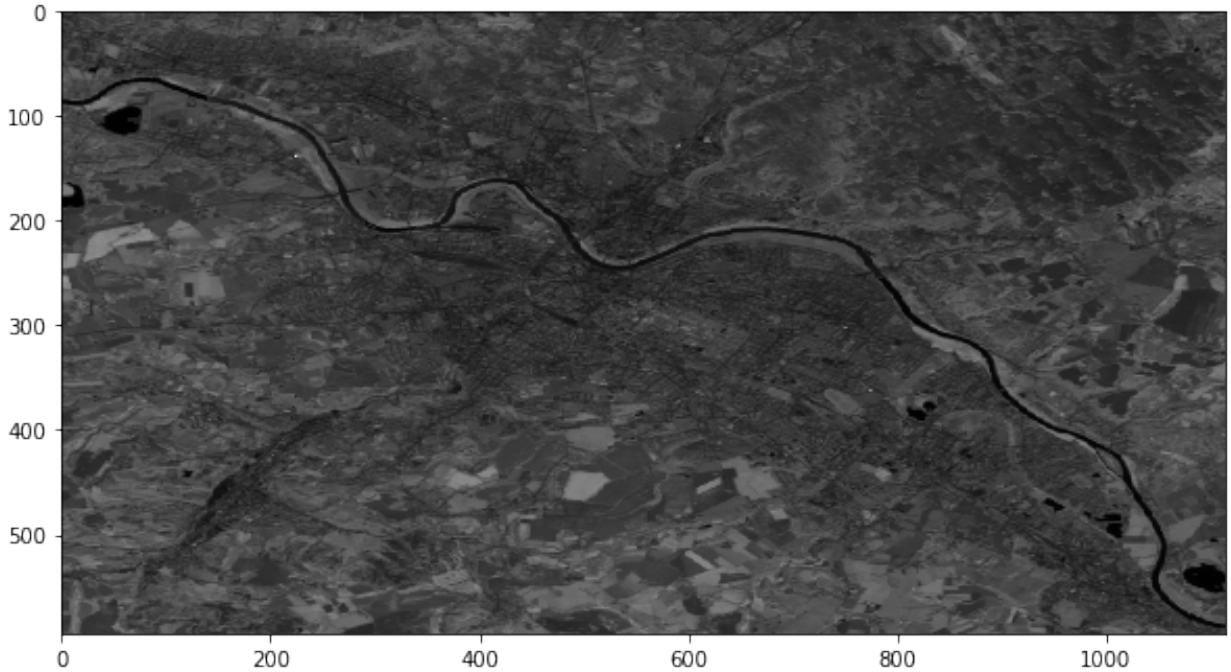
```
from climada.util import DEMO_DIR

swiss_pop = DEMO_DIR.joinpath("earth_engine", "population-density_median.tif")
dresden = DEMO_DIR.joinpath("earth_engine", "dresden.tif")  # B4 of Dresden example
```

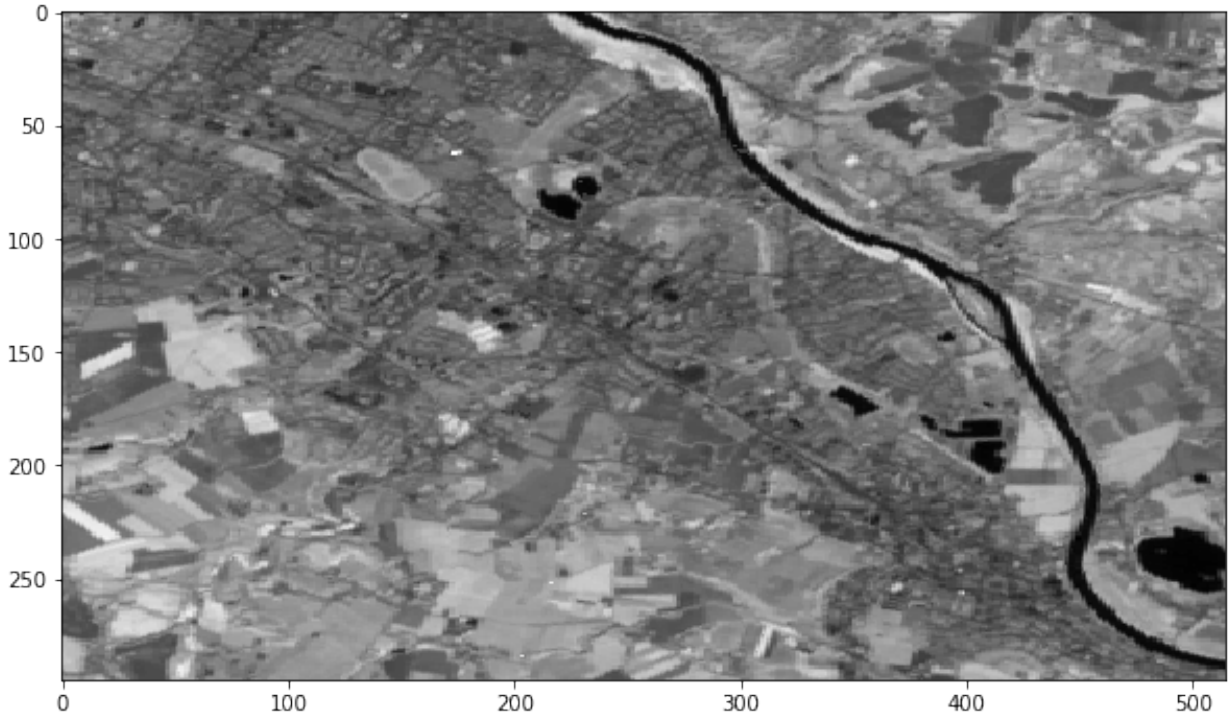
```
# Read a tif in python and Visualize the image
image_dresden = imread(dresden)
plt.figure(figsize=(10, 10))
plt.imshow(image_dresden, cmap="gray", interpolation="nearest")
plt.axis()
plt.show()

# Crop the image
image_dresden_crop = image_dresden[300:700, 600:1400]
plt.figure(figsize=(10, 10))
plt.imshow(image_dresden_crop, cmap="gray", interpolation="nearest")
```

(continues on next page)

(continued from previous page)

```
plt.axis()
plt.show()
```



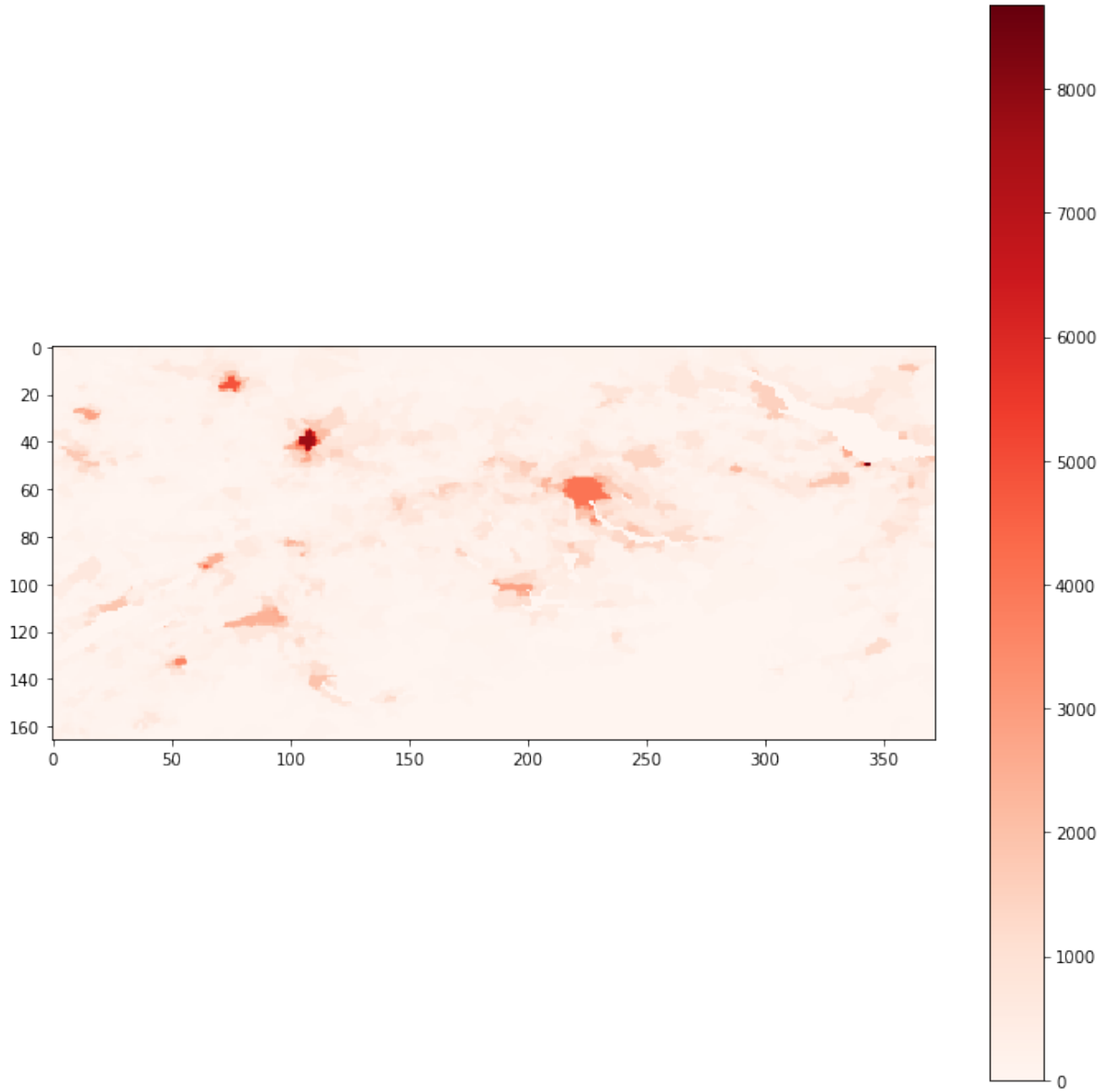



```
image_pop = imread(swiss_pop)
plt.figure(figsize=(12, 12))
plt.imshow(image_pop, cmap="Reds", interpolation="nearest")
```

(continues on next page)

(continued from previous page)

```
plt.colorbar()
plt.axis()
plt.show()
```

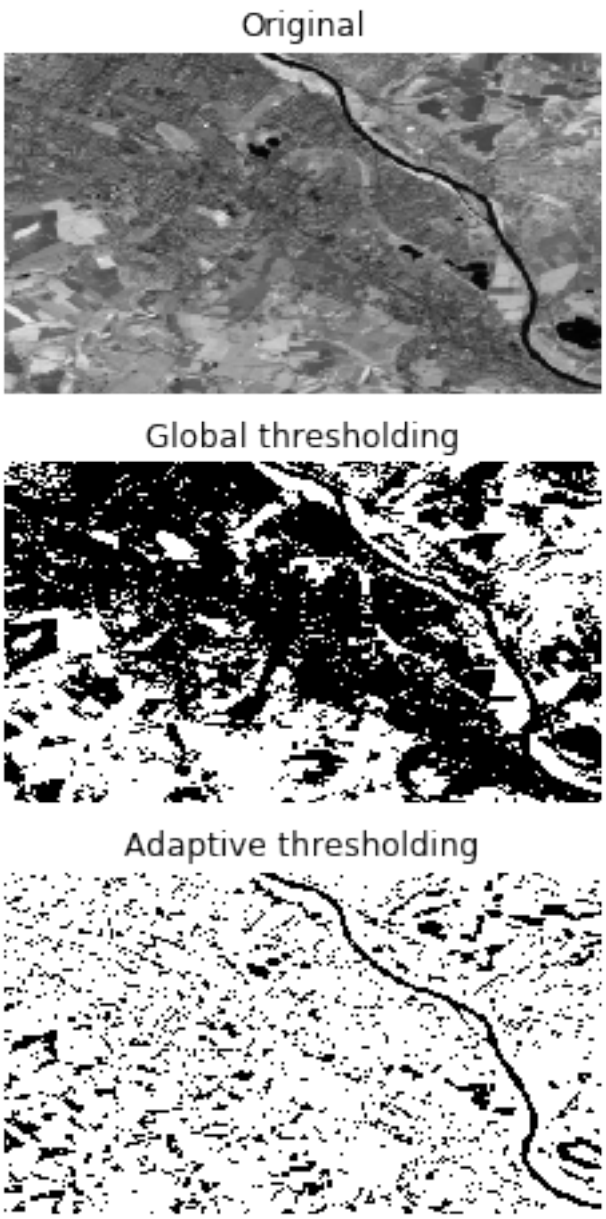



```
# Thresholding: Selection of pixels with regards with their value

global_thresh = threshold_otsu(image_dresden_crop)
binary_global = image_dresden_crop > global_thresh

block_size = 35
adaptive_thresh = threshold_local(image_dresden_crop, block_size, offset=10)
binary_adaptive = image_dresden_crop > adaptive_thresh
```

(continues on next page)

(continued from previous page)

```
fig, axes = plt.subplots(nrows=3, figsize=(7, 8))
ax = axes.ravel()
plt.gray()

ax[0].imshow(image_dresden_crop)
ax[0].set_title("Original")

ax[1].imshow(binary_global)
ax[1].set_title("Global thresholding")

ax[2].imshow(binary_adaptive)
ax[2].set_title("Adaptive thresholding")

for a in ax:
    a.axis("off")
plt.show()

print(np.sum(binary_global))
```




```
64832
```

2.11 Data API

This tutorial is separated into three main parts: the first two parts shows how to find and get data to do impact calculations and should be enough for most users. The third part provides more detailed information on how the API is built.

2.11.1 Finding datasets

```
from climada.util.api_client import Client

client = Client()
```

Data types and data type groups

The datasets are first separated into 'data_type_groups', which represent the main classes of CLIMADA (exposures, hazard, vulnerability, …). So far, data is available for exposures and hazard. Then, data is separated into data_types, representing the different hazards and exposures available in CLIMADA

```
import pandas as pd

data_types = client.list_data_type_infos()

dtf = pd.DataFrame(data_types)
dtf.sort_values(["data_type_group", "data_type"])
```

```
                 data_type data_type_group  status  \
6                centroids       centroids  active
4           crop_production       exposures  active
10                    crops       exposures  active
0                   litpop       exposures  active
13  aqueduct_coastal_flood          hazard  active
8               earthquake          hazard  active
9                    flood          hazard  active
12                    hail          hazard  active
7       relative_cropyield          hazard  active
3              river_flood          hazard  active
5             storm_europe          hazard  active
1         tropical_cyclone          hazard  active
2                 wildfire          hazard  active
11       hail_damage_crops          impact  active

                                          description  \
6                                                None
4   Historical and twenty-first century crop produ...
10  Crop exposure for Switzerland in 2021 obtained...
0   A global high-resolution asset exposure datase...
13  Global probabilistic coastal flood maps for ba...
8   Earthquake hazard sets at 150 arcsec (ca. 4km)...
9   Flood footprint of historical events at a 200m...
12  Radar-based daily hail hazard data at 1km spat...
7   Historical and twenty-first century crop produ...
3   River flood [flood depth in meters and flooded...
5   European winter storm [gust in meters per seco...
1   Tropical cyclone wind footprints (m/s) at 150 ...
2   Global wildfire dataset at 4km resolution, bas...
11  Hail damage to crops for 10-12 hail days (depe...

                                           properties  \
6   [{'property': 'res_arcsec_land', 'mandatory': ...
4   [{'property': 'crop', 'mandatory': True, 'desc...
10  [{'property': 'crop', 'mandatory': True, 'desc...
0   [{'property': 'spatial_coverage', 'mandatory':...
13  [{'property': 'climate_scenario', 'mandatory':...
8   [{'property': 'res_arcsec', 'mandatory': True,...
9   [{'property': 'date_creation', 'mandatory': Tr...
12  [{'property': 'variable', 'mandatory': True, '...
```

(continues on next page)

(continued from previous page)

```
7   [{'property': 'climate_scenario', 'mandatory':...
3   [{'property': 'res_arcsec', 'mandatory': True,...
5   [{'property': 'data_source', 'mandatory': Fals...
1   [{'property': 'spatial_coverage', 'mandatory':...
2   [{'property': 'res_arcsec', 'mandatory': True,...
11  [{'property': 'crop', 'mandatory': True, 'desc...

                                          key_reference  \
6                                                    []
4   [{'ref_no': 1, 'ref_url': 'https://doi.org/10....
10  [{'ref_no': 1, 'ref_url': 'https://doi.org/10....
0   [{'ref_no': 1, 'ref_url': 'https://doi.org/10....
13  [{'ref_no': 1, 'ref_url': 'https://www.wri.org...
8   [{'ref_no': 1, 'ref_url': None, 'key_reference...
9                                                    []
12  [{'ref_no': 1, 'ref_url': 'https://doi.org/10....
7   [{'ref_no': 1, 'ref_url': 'https://doi.org/10....
3   [{'ref_no': 1, 'ref_url': 'https://www.nature....
5   [{'ref_no': 1, 'ref_url': 'https://www.nat-haz...
1   [{'ref_no': 1, 'ref_url': 'https://doi.org/10....
2   [{'ref_no': 1, 'ref_url': 'https://gmd.coperni...
11  [{'ref_no': 1, 'ref_url': 'https://doi.org/10....

                                          version_notes
6                                                    []
4                                                    []
10                                                   []
0   [{'version': 'v2', 'notes': 'Countries with no...
13                                                   []
8                                                    []
9                                                    []
12                                                   []
7                                                    []
3   [{'version': 'v2', 'notes': 'Higher resolution...
5                                                    []
1   [{'version': 'v2', 'notes': 'Higher resolution...
2                                                    []
11                                                   []
```

Datasets and Properties

For each data type, the single datasets can be differentiated based on properties. The following function provides a table listing the properties and possible values. This table does not provide information on properties that can be combined but the search can be refined in order to find properties to query a unique dataset. Note that a maximum of 10 property values are shown here, but many more countries are available for example.

```
litpop_dataset_infos = client.list_dataset_infos(data_type="litpop")
```

```
all_properties = client.get_property_values(litpop_dataset_infos)
```

```
all_properties.keys()
```

```
dict_keys(['res_arcsec', 'exponents', 'fin_mode', 'spatial_coverage', 'country_
↪iso3alpha', 'country_name', 'country_iso3num'])
```

Refining the search:

```
# as datasets are usually available per country, chosing a country or global dataset␣
↪reduces the options
# here we want to see which datasets are available for litpop globally:
client.get_property_values(
    litpop_dataset_infos, known_property_values={"spatial_coverage": "global"}
)
```

```
{'res_arcsec': ['150'],
 'exponents': ['(0,1)', '(1,1)', '(3,0)'],
 'fin_mode': ['pop', 'pc'],
 'spatial_coverage': ['global']}
```

```
# and here for Switzerland:
client.get_property_values(
    litpop_dataset_infos, known_property_values={"country_name": "Switzerland"}
)
```

```
{'res_arcsec': ['150'],
 'exponents': ['(3,0)', '(0,1)', '(1,1)'],
 'fin_mode': ['pc', 'pop'],
 'spatial_coverage': ['country'],
 'country_iso3alpha': ['CHE'],
 'country_name': ['Switzerland'],
 'country_iso3num': ['756']}
```

2.11.2 Basic impact calculation

We here show how to make a basic impact calculation with tropical cyclones for Haiti, for the year 2040, rcp4.5 and generated with 10 synthetic tracks. For more technical details on the API, see below.

Wrapper functions to open datasets as CLIMADA objects

The wrapper functions client.get_hazard()

gets the dataset information, downloads the data and opens it as a hazard instance

```
tc_dataset_infos = client.list_dataset_infos(data_type="tropical_cyclone")
client.get_property_values(
    tc_dataset_infos, known_property_values={"country_name": "Haiti"}
)
```

```
{'res_arcsec': ['150'],
 'event_type': ['synthetic', 'observed'],
 'model_name': ['random_walk', 'STORM'],
 'spatial_coverage': ['country'],
 'climate_scenario': ['rcp26', 'rcp45', 'None', 'rcp60', 'rcp85'],
```

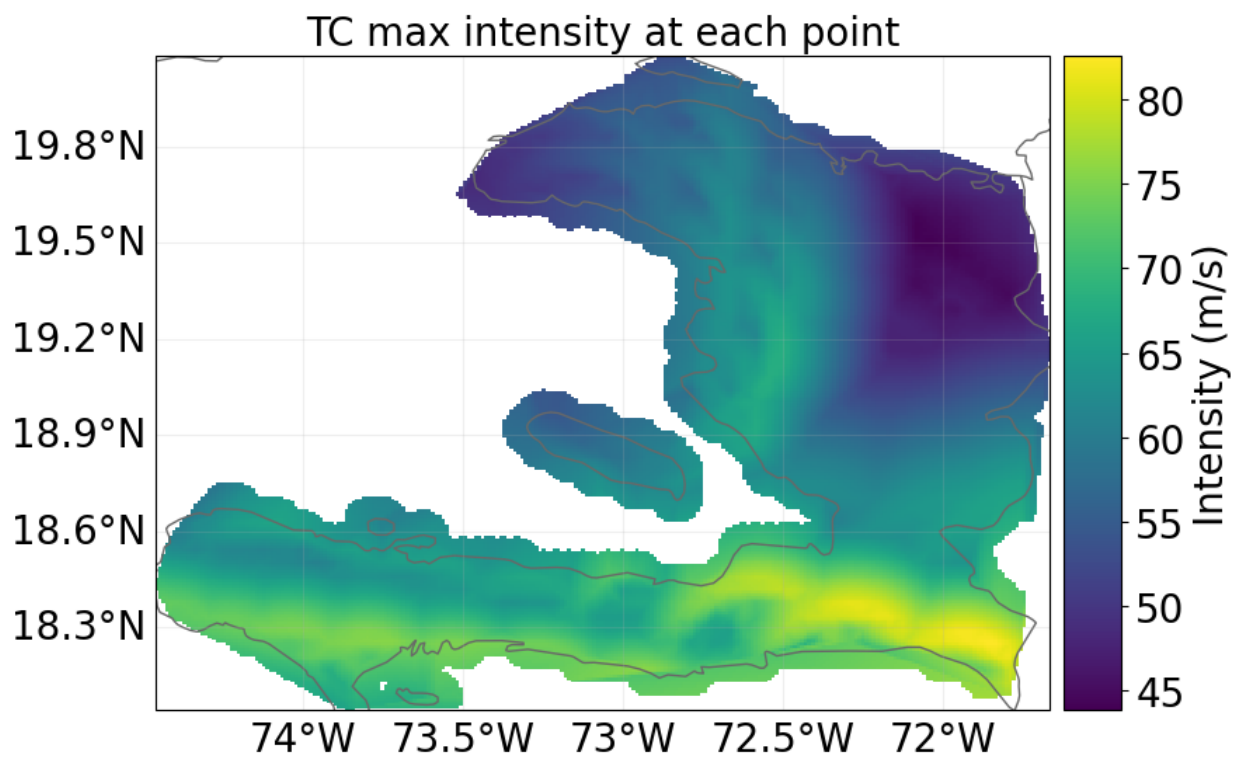
(continues on next page)

(continued from previous page)

```
 'ref_year': ['2040', '2060', '2080'],
 'country_iso3alpha': ['HTI'],
 'country_name': ['Haiti'],
 'country_iso3num': ['332'],
 'gcm': ['CMCC-CM2-VHR4', 'CNRM-CM6-1-HR', 'EC-Earth3P-HR', 'HadGEM3-GC31-HM']}
```

```
client = Client()
tc_haiti = client.get_hazard(
    "tropical_cyclone",
    properties={
        "country_name": "Haiti",
        "climate_scenario": "rcp45",
        "ref_year": "2040",
        # "nb_synth_tracks": "10",
    },
)
tc_haiti.plot_intensity(0);
```

```
2025-07-11 11:19:49,740 - climada.hazard.io - INFO - Reading C:\Users\timschmi\
↪climada\data\hazard\tropical_cyclone\tropical_cyclone_10synth_tracks_150arcsec_
↪rcp45_HTI_2040\v2.1\tropical_cyclone_10synth_tracks_150arcsec_rcp45_HTI_2040.hdf5
```



The wrapper functions client.get_litpop()

gets the default litpop, with exponents (1,1) and 'produced capital' as financial mode. If no country is given, the global dataset will be downloaded.

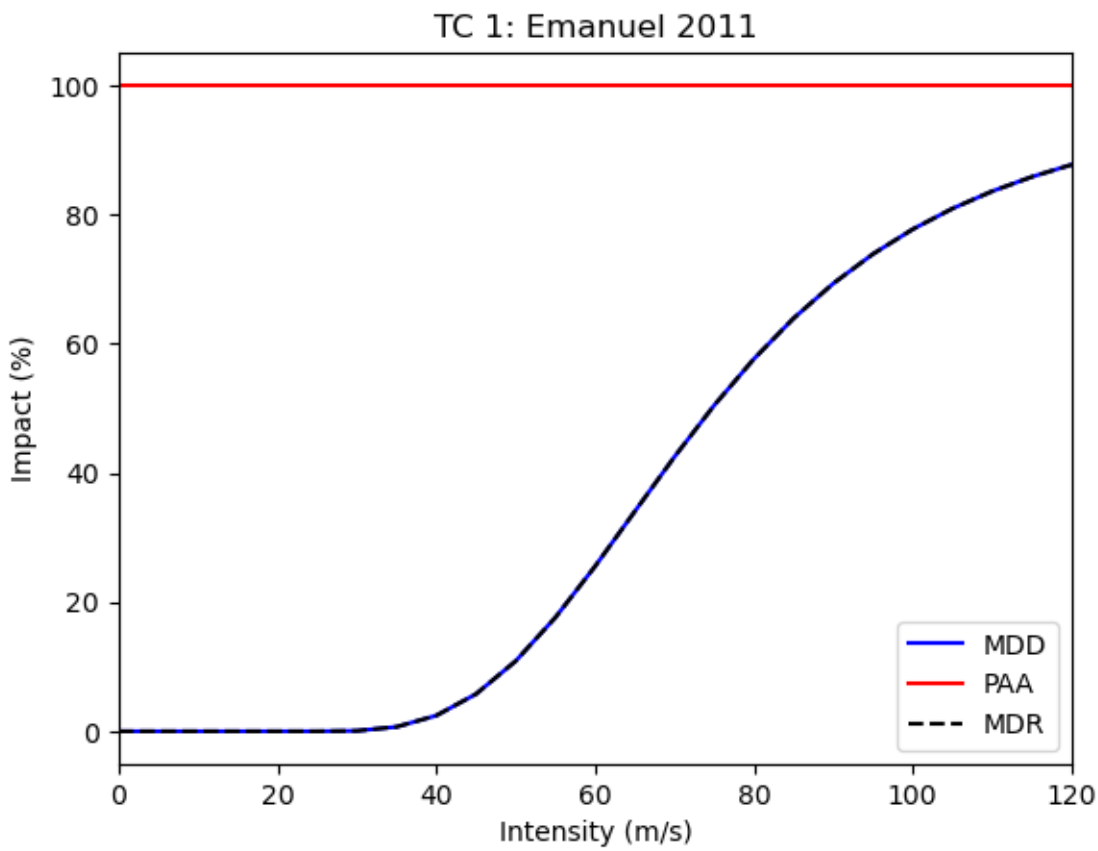
```
litpop_default = client.get_property_values(
    litpop_dataset_infos, known_property_values={"fin_mode": "pc", "exponents": "(1,1)
↪"}
)
```

```
litpop = client.get_litpop(country="Haiti")
```

```
2025-07-04 14:36:24,838 - climada.entity.exposures.base - INFO - Reading C:\Users\
↪timschmi\climada\data\exposures\litpop\LitPop_150arcsec_HTI\v3\LitPop_150arcsec_HTI.
↪hdf5
```

Get the default impact function for tropical cyclones

```
from climada.entity.impact_funcs import ImpactFuncSet, ImpfTropCyclone

imp_fun = ImpfTropCyclone.from_emanuel_usa()
imp_fun.check()
imp_fun.plot()

imp_fun_set = ImpactFuncSet([imp_fun])

litpop.impact_funcs = imp_fun_set
```



Calculate the impact

```
from climada.engine import ImpactCalc

impact = ImpactCalc(litpop, imp_fun_set, tc_haiti).impact()
```

```
2025-07-04 14:36:34,432 - climada.entity.exposures.base - INFO - No specific impact␣
↪function column found for hazard TC. Using the anonymous 'impf_' column.
2025-07-04 14:36:34,432 - climada.entity.exposures.base - INFO - Matching 1329␣
↪exposures with 1332 centroids.
2025-07-04 14:36:34,432 - climada.util.coordinates - INFO - No exact centroid match␣
↪found. Reprojecting coordinates to nearest neighbor closer than the threshold = 100
2025-07-04 14:36:34,432 - climada.engine.impact_calc - INFO - Calculating impact for␣
↪3987 assets (>0) and 43560 events.
```

Getting other Exposures

```
crop_dataset_infos = client.list_dataset_infos(data_type="crop_production")

client.get_property_values(crop_dataset_infos)
```

```
{'crop': ['whe', 'soy', 'ric', 'mai'],
 'irrigation_status': ['noirr', 'firr'],
```

(continues on next page)

(continued from previous page)

```
 'unit': ['USD', 'Tonnes'],
 'spatial_coverage': ['global']}
```

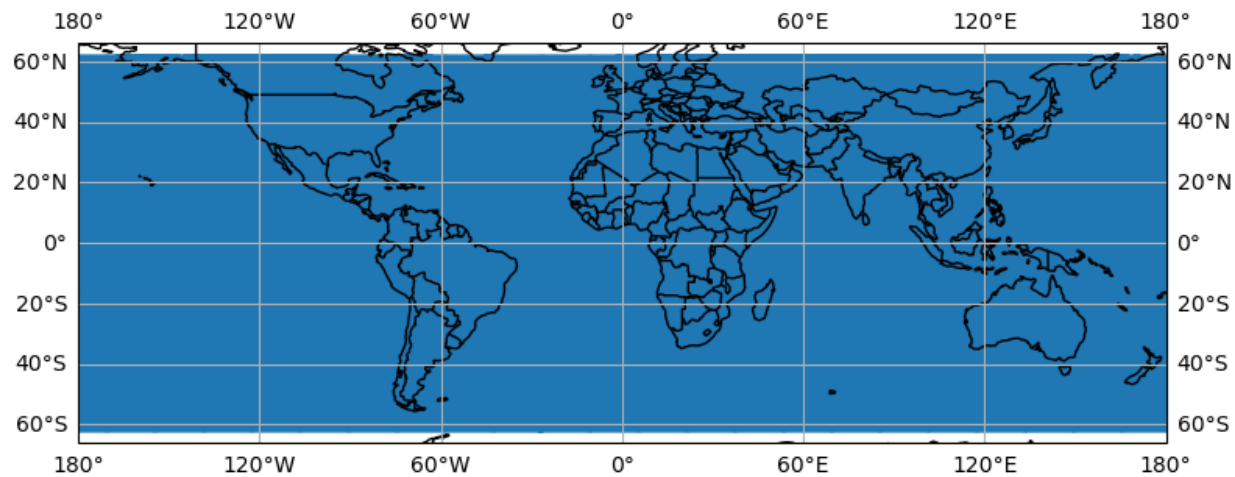
```
rice_exposure = client.get_exposures(
    exposures_type="crop_production",
    properties={"crop": "ric", "unit": "USD", "irrigation_status": "noirr"},
)
```

```
2025-07-04 14:36:49,797 - climada.entity.exposures.base - INFO - Reading C:\Users\
↪timschmi\climada\data\exposures\crop_production\crop_production_ric-noirr_spamray-
↪mirca_USD\v1\crop_production_ric-noirr_spamray-mirca_USD.hdf5
```

Getting base centroids to generate new hazard files

```
centroids = client.get_centroids()
centroids.plot()
```

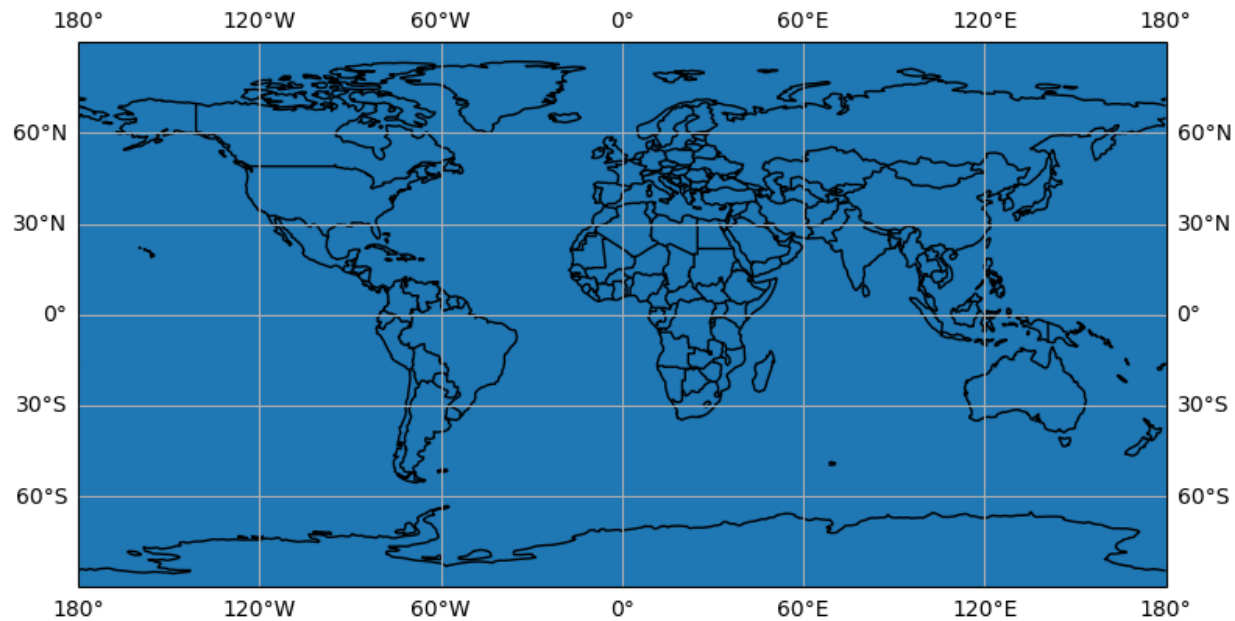
```
<GeoAxes: >
```



By default the latitude extent is limited to [-60,60], which is sufficient for many hazards and will reduce the computational ressources required. If a complete latitude extent [-90,90] or a smaller extent is desired, it can be specified with the `extent` keyword agrument.

```
centroids_complete = client.get_centroids(extent=[-180, 180, -90, 90])
centroids_complete.plot()
```

```
<GeoAxes: >
```



centroids are also available per country:

```
centroids_hti = client.get_centroids(country="HTI")
```

2.11.3 Technical Information

For programmatical access to the CLIMADA data API there is a specific REST call wrapper class: `climada.util.client.Client`.

Server

The CLIMADA data file server is hosted on https://data.iac.ethz.ch that can be accessed via a REST API at https://climada.ethz.ch. For REST API details, see the documentation.

Client

```
?Client
```

```
Init signature: Client(cache_enabled=None)
Docstring:      Python wrapper around REST calls to the CLIMADA data API server.
Init docstring:
Constructor of Client.

Data API host and chunk_size (for download) are configurable values.
Default values are 'climada.ethz.ch' and 8096 respectively.

Parameters
----------
cache_enabled : bool, optional
    This flag controls whether the api calls of this client are going to be cached to␣
↪the
    local file system (location defined by CONFIG.data_api.cache_dir).
```

(continues on next page)

(continued from previous page)

```
    If set to true, the client can reload the results from the cache in case there is␣
↪no
    internet connection and thus work in offline mode.
    Default: None, in this case the value is taken from CONFIG.data_api.cache_enabled.
File:           c:\users\timschmi\documents\phd\code\climada_python\climada\util\api_
↪client.py
Type:           type
Subclasses:
```

```
client = Client()
client.chunk_size
```

```
8192
```

The url to the API server and the chunk size for the file download can be configured in 'climada.conf'. Just replace the corresponding default values:

```
    "data_api": {
        "host": "https://climada.ethz.ch",
        "chunk_size": 8192,
        "cache_db": "{local_data.system}/.downloads.db"
    }
```

The other configuration value affecting the data_api client, `cache_db`, is the path to an SQLite database file, which is keeping track of the files that are successfully downloaded from the api server. Before the Client attempts to download any file from the server, it checks whether the file has been downloaded before and if so, whether the previously downloaded file still looks good (i.e., size and time stamp are as expected). If all of this is the case, the file is simply read from disk without submitting another request.

Metadata

Unique Identifiers

Any dataset can be identified with **data_type**, **name** and **version**. The combination of the three is unique in the API servers' underlying database. However, sometimes the name is already enough for identification. All datasets have a UUID, a universally unique identifier, which is part of their individual url. E.g., the uuid of the dataset https://climada.ethz.ch/rest/dataset/b1c76120-4e60-4d8f-99c0-7e1e7b7860ec is "b1c76120-4e60-4d8f-99c0-7e1e7b7860ec". One can retrieve their meta data by:

```
client.get_dataset_info_by_uuid("b1c76120-4e60-4d8f-99c0-7e1e7b7860ec")
```

```
DatasetInfo(uuid='b1c76120-4e60-4d8f-99c0-7e1e7b7860ec', data_
↪type=DataTypeShortInfo(data_type='litpop', data_type_group='exposures'), name=
↪'LitPop_assets_pc_150arcsec_SGS', version='v1', status='expired', properties={'res_
↪arcsec': '150', 'exponents': '(3,0)', 'fin_mode': 'pc', 'spatial_coverage': 'country
↪', 'date_creation': '2021-09-23', 'climada_version': 'v2.2.0', 'country_iso3alpha':
↪'SGS', 'country_name': 'South Georgia and the South Sandwich Islands', 'country_
↪iso3num': '239'}, files=[FileInfo(uuid='b1c76120-4e60-4d8f-99c0-7e1e7b7860ec', url=
↪'https://data.iac.ethz.ch/climada/b1c76120-4e60-4d8f-99c0-7e1e7b7860ec/LitPop_
↪assets_pc_150arcsec_SGS.hdf5', file_name='LitPop_assets_pc_150arcsec_SGS.hdf5',␣
↪file_format='hdf5', file_size=1086488, check_sum=
↪'md5:27bc1846362227350495e3d946dfad5e')], doi=None, description="LitPop asset value␣
```

(continues on next page)

(continued from previous page)

```
↪exposure per country: Gridded physical asset values by country, at a resolution of␣
↪150 arcsec. Values are total produced capital values disaggregated proportionally␣
↪to the cube of nightlight intensity (Lit^3, based on NASA Earth at Night). The␣
↪following values were used as parameters in the LitPop.from_countries() method:{
↪'total_values': 'None', 'admin1_calc': 'False','reference_year': '2018', 'gpw_
↪version': '4.11'}Reference: Eberenz et al., 2020. https://doi.org/10.5194/essd-12-
↪817-2020", license='Attribution 4.0 International (CC-BY-4.0)', activation_date=
↪'2021-09-13 09:08:28.358559+00:00', expiration_date='2022-06-27 14:35:55.
↪203081+00:00')
```

or by filtering:

Data Set Status

The datasets of climada.ethz.ch may have the following stati:

- **active**: the default for real life data
- **preliminary**: when the dataset is already uploaded but some information or file is still missing
- **expired**: when a dataset is inactivated again
- **test_dataset**: data sets that are used in unit or integration tests have this status in order to be taken seriously by accident When collecting a list of datasets with `get_datasets`, the default dataset status will be 'active'. With the argument `status=None` this filter can be turned off.

DatasetInfo Objects and DataFrames

As stated above `get_dataset` (or `get_dataset_by_uuid`) return a `DatasetInfo` object and `get_datasets` a list thereof.

```
from climada.util.api_client import DatasetInfo

?DatasetInfo
```

```
Init signature:
DatasetInfo(
    uuid: str,
    data_type: climada.util.api_client.DataTypeShortInfo,
    name: str,
    version: str,
    status: str,
    properties: dict,
    files: list,
    doi: str,
    description: str,
    license: str,
    activation_date: str,
    expiration_date: str,
) -> None
Docstring:      dataset data from CLIMADA data API.
File:           c:\users\timschmi\documents\phd\code\climada_python\climada\util\api_
↪client.py
```

(continues on next page)

(continued from previous page)

```
Type:           type
Subclasses:
```

where files is a list of `FileInfo` objects:

```
from climada.util.api_client import FileInfo

?FileInfo
```

```
Init signature:
FileInfo(
    uuid: str,
    url: str,
    file_name: str,
    file_format: str,
    file_size: int,
    check_sum: str,
) -> None
Docstring:      file data from CLIMADA data API.
File:           c:\users\timschmi\documents\phd\code\climada_python\climada\util\api_
↪client.py
Type:           type
Subclasses:
```

Convert into DataFrame

There is a conveinience function to easily convert dataset lists into pandas DataFrames, `into_datasets_df`:

```
?client.into_datasets_df
```

```
Signature: client.into_datasets_df(dataset_infos)
Docstring:
Convenience function providing a DataFrame of datasets with properties.

Parameters
----------
dataset_infos : list of DatasetInfo
     as returned by list_dataset_infos

Returns
-------
pandas.DataFrame
    of datasets with properties as found in query by arguments
File:      c:\users\timschmi\documents\phd\code\climada_python\climada\util\api_
↪client.py
Type:      function
```

```
from climada.util.api_client import Client

client = Client()
litpop_datasets = client.list_dataset_infos(
```

(continues on next page)

(continued from previous page)

```
    data_type="litpop",
    properties={"country_name": "Germany"},
)
litpop_df = client.into_datasets_df(litpop_datasets)
litpop_df
```

```
  data_type data_type_group                                  uuid  \
0    litpop       exposures  5a3e72fb-a8e0-48b5-b2ec-d6d9b845b687
1    litpop       exposures  552ce473-953f-44db-9504-0bf078ba7cb7
2    litpop       exposures  ca0b1390-0ccc-4bac-8867-c6f9c0ef7703

                            name version  status   doi  \
0  LitPop_assets_pc_150arcsec_DEU      v3  active  None
1        LitPop_pop_150arcsec_DEU      v3  active  None
2            LitPop_150arcsec_DEU      v3  active  None

                                        description  \
0  LitPop asset value exposure per country: Gridd...
1  LitPop population exposure per country: Gridde...
2  LitPop asset value exposure per country: Gridd...

                                     license  \
0  Attribution 4.0 International (CC-BY-4.0)
1  Attribution 4.0 International (CC-BY-4.0)
2  Attribution 4.0 International (CC-BY-4.0)

                    activation_date expiration_date res_arcsec exponents  \
0  2024-06-07 08:21:45.775654+00:00            None        150     (3,0)
1  2024-06-07 08:21:51.998009+00:00            None        150     (0,1)
2  2024-06-07 08:21:55.563907+00:00            None        150     (1,1)

  fin_mode spatial_coverage  climada_version date_creation country_iso3alpha  \
0       pc          country  v4.1.1 (v3.1.2)     2022-6-30               DEU
1      pop          country  v4.1.1 (v3.1.2)     2022-6-30               DEU
2       pc          country  v4.1.1 (v3.1.2)     2022-6-26               DEU

  country_name country_iso3num
0      Germany             276
1      Germany             276
2      Germany             276
```

Download

The wrapper functions `get_exposures` or `get_hazard` fetch the information, download the file and opens the file as a climada object. But one can also just download dataset files using the method `download_dataset` which takes a `DatasetInfo` object as argument and downloads all files of the dataset to a directory in the local file system.

```
?client.download_dataset
```

```
Signature:
client.download_dataset(
```

(continues on next page)

(continued from previous page)

```
    dataset,
    target_dir=WindowsPath('C:/Users/timschmi/climada/data'),
    organize_path=True,
)
Docstring:
Download all files from a given dataset to a given directory.

Parameters
----------
dataset : DatasetInfo
    the dataset
target_dir : Path, optional
    target directory for download, by default `climada.util.constants.SYSTEM_DIR`
organize_path: bool, optional
    if set to True the files will end up in subdirectories of target_dir:
    [target_dir]/[data_type_group]/[data_type]/[name]/[version]
    by default True

Returns
-------
download_dir : Path
    the path to the directory containing the downloaded files,
    will be created if organize_path is True
downloaded_files : list of Path
    the downloaded files themselves

Raises
------
Exception
    when one of the files cannot be downloaded
File:      c:\users\timschmi\documents\phd\code\climada_python\climada\util\api_
↪client.py
Type:      method
```

Cache

The method avoids superfluous downloads by keeping track of all downloads in a sqlite db file. The client will make sure that the same file is never downloaded to the same target twice.

Examples

```
# Let's have a look at an example for downloading a litpop dataset first
ds = litpop_datasets[
    0
]  # litpop_datasets is a list and download_dataset expects a single object as␣
↪argument.
download_dir, ds_files = client.download_dataset(ds)
ds_files[0], ds_files[0].is_file()
```

```
(WindowsPath('C:/Users/timschmi/climada/data/exposures/litpop/LitPop_assets_pc_
↪150arcsec_DEU/v3/LitPop_assets_pc_150arcsec_DEU.hdf5'),
```

(continues on next page)

(continued from previous page)

```
 True)
```

```
# Another example for downloading a hazard (tropical cyclone) dataset
ds_tc = tc_dataset_infos[0]
download_dir, ds_files = client.download_dataset(ds_tc)
ds_files[0], ds_files[0].is_file()
```

```
(WindowsPath('C:/Users/timschmi/climada/data/hazard/tropical_cyclone/tropical_cyclone_
↪10synth_tracks_150arcsec_rcp26_global_2040/v2.1/tropical_cyclone_10synth_tracks_
↪150arcsec_rcp26_global_2040.hdf5'),
 True)
```

If the dataset contains only one file (which is most commonly the case) this file can also be downloaded and accessed in a single step, using the `get_dataset_file` method:

```
from climada.util.api_client import Client

Client().get_dataset_file(
    data_type="litpop",
    properties={
        "country_name": "South Georgia and the South Sandwich Islands",
        "fin_mode": "pop",
    },
)
```

```
WindowsPath('C:/Users/timschmi/climada/data/exposures/litpop/LitPop_pop_150arcsec_SGS/
↪v3/LitPop_pop_150arcsec_SGS.hdf5')
```

Local File Cache

By default, the API Client downloads files into the `~/climada/data` directory.

In the course of time obsolete files may be accumulated within this directory, because there is a newer version of these files available from the CLIMADA data API, or because the according dataset got expired altogether.
To prevent file rot and free disk space, it's possible to remove all outdated files at once, by simply calling `Client().purge_cache()`. This will remove all files that were ever downloaded with the `api_client.Client` and for which a newer version exists, even when the newer version has not been downloaded yet.

Offline Mode

The API Client is silently used in many methods and functions of CLIMADA, including the installation test that is run to see whether the CLIMADA installation was successful. Most methods of the client send GET requests to the API server assuming the latter is accessible through a working internet connection. If this is not the case, the functionality of CLIMADA is severely limited if not altogether lost. Often this is an unnecessary restriction, e.g., when a user wants to access a file through the API Client that is already downloaded and available in the local filesystem.

In such cases the API Client runs in *offline mode*. In this mode the client falls back to previous results for the same call in case there is no internet connection or the server is not accessible.

To turn this feature off and make sure that all results are current and up to date - at the cost of failing when there is no internet connection - one has to disable tha *cache*. This can be done programmatically, by initializing the API Client with the optional argument `cache_enabled`:

```
client = Client(cache_enabled=False)
```

Or it can be done through configuration. Edit the `climada.conf` file in the working directory or in ~/climada/ and change the "cache_enabled" value, like this:

```
...
    "data_api": {
        ...
        "cache_enabled": false
    },
...
```

While `cache_enabled` is `true` (default), every result from the server is stored as a json file in ~/climada/data/.apicache/ by a unique name derived from the method and arguments of the call. If the very same call is made again later, at a time where the server is not accessible, the client just comes back to the cached result from the previous call.

2.12 Citation Guide

If you use CLIMADA for your work, please cite the appropriate publications. A list of all CLIMADA code related articles is available on Zotero and can be downloaded as single Bibtex file: `climada_publications.bib`

2.12.1 Publications by Module

If you use specific tools and modules of CLIMADA, please cite the appropriate publications presenting these modules according to the following table:

Module or tool used	Publication to cite
Any	The Zenodo archive of the CLIMADA version you are using
Impact calculations	Aznar-Siguan, G. and Bresch, D. N. (2019): CLIMADA v1: A global weather and climate risk assessment platform, Geosci. Model Dev., 12, 3085–3097, https://doi.org/10.5194/gmd-12-3085-2019
Cost-benefit analysis	Bresch, D. N. and Aznar-Siguan, G. (2021): CLIMADA v1.4.1: Towards a globally consistent adaptation options appraisal tool, Geosci. Model Dev., 14, 351–363, https://doi.org/10.5194/gmd-14-351-2021
Uncertainty and sensitivity analysis	Kropf, C. M. et al. (2022): Uncertainty and sensitivity analysis for probabilistic weather and climate-risk modelling: an implementation in CLIMADA v.3.1.0. Geosci. Model Dev. 15, 7177–7201, https://doi.org/10.5194/gmd-15-7177-2022
Lines and polygons exposures or Open Street Map exposures	Mühlhofer, E., et al. (2024): OpenStreetMap for Multi-Faceted Climate Risk Assessments : Environ. Res. Commun. 6 015005, https://doi.org/10.1088/2515-7620/ad15ab
LitPop exposures	Eberenz, S., et al. (2020): Asset exposure data for global physical risk assessment. Earth System Science Data 12, 817–833, https://doi.org/10.3929/ethz-b-000409595
Impact function calibration	Riedel, L., et al. (2024): A Module for Calibrating Impact Functions in the Climate Risk Modeling Platform CLIMADA. Journal of Open Source Software, 9(99), 6755, https://doi.org/10.21105/joss.06755
GloFAS River Flood Module	Riedel, L. et al. (2024): Fluvial flood inundation and socio-economic impact model based on open data, Geosci. Model Dev., 17, 5291–5308, https://doi.org/10.5194/gmd-17-5291-2024

Please find the code to reprocduce selected CLIMADA-related scientific publications in our repository of scientific publications.

2.12.2 Links and Logo

In presentations or other graphical material, as well as in reports etc., where applicable, please add the following logo: `climada_logo_QR.png`:

As key link, please use https://wcr.ethz.ch/research/climada.html, as it provides a brief introduction especially for those not familiar with GitHub.

CHAPTER

THREE

DEVELOPER GUIDE

This developer guide regroups all information intended for contributors.

Very minimal instruction for contributing can be found below.

If you are interested in contributing to CLIMADA, we recommand you to start with the *Overview* part of this guide.

3.1 What Warrants a Contribution?

Anything! For orientation, these are some categories of possible contributions we can think of:

- **Technical problems and bugs:** Did you encounter a problem when using CLIMADA? Raise an issue in our repository, providing a description or ideally a code replicating the error. Did you already find a solution to the problem? Please raise a pull request to help us resolve the issue!
- **Documentation and Tutorial Updates:** Found a typo in the documentation? Is a tutorial lacking some information you find important? Simply fix a line, or add a paragraph. We are happy to incorporate you additions! Please raise a pull request!
- **New Modules and Utility Functions:** Did you create a function or an entire module you find useful for your work? Maybe you are not the only one! Feel free to simply raise a pull request for functions that improve, e.g., plotting or data handling. As an entire module has to be carefully integrated into the framework, it might help if you talk to us first so we can design the module and plan the next steps. You can do that by raising an issue or starting a discussion on GitHub.

A good place to start a personal discussion is our monthly CLIMADA developers call. Please contact the lead developers if you want to join.

3.2 Why Should You Contribute?

- You will be listed as author of the CLIMADA repository in the AUTHORS file.
- You will improve the quality of the CLIMADA software for you and for everybody else using it.
- You will gain insights into scientific software development.

3.3 Minimal Steps to Contribute

Before you start, please have a look at our Developer Guide section in the CLIMADA Docs.

To contribute follow these steps:

1. Install CLIMADA following the installation instructions for developers.
2. In the CLIMADA repository, create a new feature branch from the latest `develop` branch:

```
git checkout develop && git pull
git checkout -b feature/my-fancy-branch
```

3. Implement your changes and commit them with meaningful and well formatted commit messages.
4. Add unit and integration tests to your code, if applicable.
5. Use Pylint for a static code analysis of your code with CLIMADA's configuration `.pylintrc`:

```
pylint
```

6. Add your name to the AUTHORS file.
7. Push your updates to the remote repository:

```
git push --set-upstream origin feature/my-fancy-branch
```

NOTE: Only team members are allowed to push to the original repository. Most contributors are/will be team members. To be added to the team list and get permissions please contact one of the owners. Alternatively, you can fork the CLIMADA repository and add this fork as a new remote to your local repository. You can then push to the fork remote:

```
git remote add fork <your-fork-url>
git push --set-upstream fork feature/my-fancy-branch
```

8. On the CLIMADA-project/climada_python GitHub repository, create a new pull request with target branch `develop`. This also works if you pushed to a fork instead of the main repository. Add a description and explanation of your changes and work through the pull request author checklist provided. Feel free to request reviews from specific team members.
9. After approval of the pull request, the branch is merged into `develop` and your changes will become part of the next CLIMADA release.

3.4 Resources

The CLIMADA documentation provides several Developer Guides. Here's a selection of the commonly required information:

- How to use Git and GitHub for CLIMADA development: Development and Git and CLIMADA
- Coding instructions for CLIMADA: Python Dos and Don'ts, Performance Tips, CLIMADA Conventions
- How to execute tests in CLIMADA: Testing and Continuous Integration

3.5 Pull Requests

After developing a new feature, fixing a bug, or updating the tutorials, you can create a pull request to have your changes reviewed and then merged into the CLIMADA code base. To ensure that your pull request can be reviewed quickly and easily, please have a look at the *Resources* above before opening a pull request. In particular, please check out the Pull Request instructions.

We provide a description template for pull requests that helps you provide the essential information for reviewers. It also contains a checklist for both pull request authors and reviewers to guide the review process.

3.6 CLIMADA Development

This is a guide about how to contribute to the development of CLIMADA. We first explain some general guidelines about when and how one can contribute to CLIMADA, and then describe the steps in detail. We assume that you are familiar with Git, Github and their commands. If you are not familiar with these, you can refer to our instructions for *Development with Git*.

3.6.1 Is CLIMADA the right place for your contribution?

When developing for CLIMADA, it is important to distinguish between core content and particular applications. Core content is meant to be included into the climada_python repository and will be subject to a code review. Any new addition should first be discussed with one of the repository admins. The purpose of this discussion is to see

- How does the planned module fit into CLIMADA?
- What is an optimal architecture for the new module?
- What parts might already exist in other parts of the code?

Applications made with CLIMADA, such as an ECA study can be stored in the paper repository once they have been published. For other types of work, consider making a separate repository that imports CLIMADA as an external package.

3.6.2 Planning a new feature

Here we're talking about large features such as new modules, new data sources, or big methodological changes. Any extension to CLIMADA that might affect other developers' work, modify the CLIMADA core, or need a big code review.

Smaller feature branches don't need such formalities. Use your judgment, and if in doubt, let people know.

Talk to the group

- Before starting coding a module, do not forget to coordinate with one of the repo admins (Emanuel, Chahan or Lukas)
- This is the chance to work out the Big Picture stuff that is better when it's planned with the group - possible intersections with other projects, possible conflicts, changes to the CLIMADA core, additional dependencies
- Also talk with others from the core development team (see the GitHub wiki).
- Bring it to a developers meeting - people may be able to help/advise and are always interested in hearing about new projects. You can also find reviewers!
- Also, keep talking! Your plans *will* change :)

Formulate the feature's data flow and workflow

To optimize implementation and usefulness of the new feature, first conceptualize its data flow and workflow. It makes sense to discuss these with a CLIMADA core developer before starting to work on the feature's implementation.

- **Data flow**: Outline of how data moves through the system — where it is created or input, how it is processed, and if and where it is stored. This helps to improve the computational efficiency and to identify potential bottlenecks.
- **Workflow**: Plan about where and how the user and other CLIMADA components can interact with the new feature. This ensures that the new feature couples seamlessly to the existing code base of CLIMADA and that the new feaute is easily and clearly accessible to users.

Planning the work

- Does the project go in its own repository and import CLIMADA, or does it extend the main CLIMADA repository. The way this is done is slowly changing, so definitely discuss it with the group.
- Find a few people who will help to review your code.
  - Ask in a developers' meeting, on Slack (for WCR developers) or message people on the development team (see the GitHub wiki).
  - Let them know roughly how much code will be in the reviews, and when you'll be creating pull requests.
- How can the work split into manageable chunks?
  - A series of smaller pull requests is far more manageable than one big one (and takes off some of the pre-release pressure)
  - Reviewing and spotting issues/improvements/generalisations early is always a good thing.
  - It encourages modularisation of the code: smaller self-contained updates, with documentation and tests.
- Will there be any changes to the CLIMADA core? These should be planned carefully
- Will you need any new dependencies? Are you sure?

3.6.3 Installing CLIMADA for development

To develop (or review a pull request), you need to setup a proper climada development environment. This is relatively easy but requires rigor, so please read all the instructions below and make sure to follow them (we also recommend to read everything once first, and then follow them from the start).

First, follow the *Advanced instructions*. Note that if you want to work on a specific branch instead of `develop`, if you work on a feature for instance), you need to checkout that specific branc instead of `develop` after cloning:

```
git clone https://github.com/CLIMADA-project/climada_python.git
cd climada_python
git checkout <other branch>
```

Note on dependencies

Climada dependencies are handled with the `requirements/env_climada.yml` file. When you run `mamba env update -n <your_env> -f requirements/env_climada.yml`, the content of that file is used to install the dependencies, thus, if you are working on a branch that changes the dependencies, make sure to be on that branch **before** running the command.

3.6.4 Working on feature branches

When developing a big new feature, consider creating a feature branch and merging smaller branches into that feature branch with pull requests, keeping the whole process separate from `develop` until it's completed. This makes step-by-step code review nice and easy, and makes the final merge more easily tracked in the history.

e.g. developing the big `feature/meteorite` module you might write `feature/meteorite-hazard` and merge it in, then `feature/meteorite-impact`, then `feature/meteorite-stochastic-events` etc… before finally merging `feature/meteorite` into `develop`. Each of these could be a reviewable pull request.

Make a new branch

For new features in Git flow:

```
git flow feature start feature_name
```

Which is equivalent to (in vanilla git):

```
git checkout -b feature/feature_name
```

Or work on an existing branch:

```
git checkout -b branch_name
```

get the latest data from the remote repository and update your branch

```
git pull
```

Once you have set up everything (including pre-commit hooks) you will be able to:

see your locally modified files

```
git status
```

add changes you want to include in the commit

```
git add climada/modified_file.py climada/test/test_modified_file.py
```

commit the changes

```
git commit -m "new functionality of .. implemented"
```

Pre-Commit Hooks

Climada developer dependencies include pre-commit hooks to help ensure code linting and formatting. See *Code Formatting* for our conventions regarding formatting. These hooks will run on all staged files and verify:

- the absence of trailing whitespace
- that files end in a newline and only a newline
- the correct sorting of imports using `isort`
- the correct formatting of the code using `black`

If you have installed the pre-commit hooks (see *Install developer dependencies*), they will be run each time you attempt to create a new commit, and the usual git flow can slightly change:

If any check fails, you will be warned and these hooks **will apply** corrections (such as formatting the code with black if it is not). As files are modified, you are required to stage them again (hooks cannot stage their modification, only you can) and commit again.

As an exemple, suppose you made an improvement to Centroids and want to commit these changes, you would run:

```
$ git status
On branch feature/<new_feature>
Your branch is up-to-date with 'origin/<new_feature>'.

Changes to be committed:
```

(continues on next page)

(continued from previous page)

```
  (use "git restore --staged <file>..." to unstage)
        modified:   climada/hazard/centroids/centr.py
```

Now trying to commit, and assuming that imports are not correctly sorted, and some of the code is not correctly formatted:

```
$ git commit -m "Add <new_feature> to centroids"
Fix End of Files.....................................................Passed
Trim Trailing Whitespace.............................................Passed
isort................................................................Failed
- hook id: isort
- files were modified by this hook

Fixing [...]/climada_python/climada/hazard/centroids/centr.py

black-jupyter........................................................Failed
- hook id: black-jupyter
- files were modified by this hook

reformatted climada/hazard/centroids/centr.py

All done! ✨ 🍰 ✨
```

Note the commit was aborted, and the problems were fixed. However, these changes added by the hooks are not *staged* yet. You have to run `git add` again to stage them:

```
$ git status
On branch feature/<new_feature>
Your branch is up-to-date with 'origin/<new_feature>'.

Changes to be committed:
  (use "git restore --staged <file>..." to unstage)
        modified:   climada/hazard/centroids/centr.py

Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   climada/hazard/centroids/centr.py

$ git add climada/hazard/centroids/centr.py
```

After that, you can execute the commit and the hooks should pass:

```
$ git commit -m "Add <new_feature> to centroids"
Fix End of Files.....................................................Passed
Trim Trailing Whitespace.............................................Passed
isort................................................................Passed
black-jupyter........................................................Passed

All done! ✨ 🍰 ✨
```

Make unit and integration tests on your code, preferably during development

Writing new code requires writing new tests: Please read our *Guide on unit and integration tests*

3.6.5 Pull requests

We want every line of code that goes into the CLIMADA repository to be reviewed!

Code review:

- catches bugs (there are *always* bugs)
- lets you draw on the experience of the rest of the team
- makes sure that more than one person knows how your code works
- helps to unify and standardise CLIMADA's code, so new users find it easier to read and navigate
- creates an archived description and discussion of the changes you've made

When to make a pull request

- When you've finished writing a big new class or method (and its tests)
- When you've fixed a bug or made an improvement you want to merge
- When you want to merge a change of code into `develop` or `main`
- When you want to *discuss* a bit of code you've been working on - pull requests aren't only for merging branches

Not all pull requests have to be into `develop` - you can make a pull request into any active branch that suits you.

Pull requests need to be made latest two weeks before a release, see releases.

Step by step pull request!

Let's suppose you've developed a cool new module on the `feature/meteorite` branch and you're ready to merge it into `develop`.

Checklist before you start

- Documentation
- Tests
- Tutorial (if a complete new feature)
- Updated dependencies (if need be)
- Added your name to the AUTHORS file
- Added an entry to the `CHANGELOG.md` file. See https://keepachangelog.com for information on how this shoud look like.
- (Advanced, optional) interactively rebase/squash recent commits that *aren't yet on GitHub*.

Steps

1) Make sure the `develop` branch is up to date on your own machine

```
git checkout develop
git pull
```

2) Merge `develop` into your feature branch and resolve any conflicts

```
git checkout feature/meteorite
git merge develop
```

In the case of more complex conflicts, you may want to speak with others who worked on the same code. Your IDE should have a tool for conflict resolution.

3) Check all the tests pass locally

```
make unit_test
make integ_test
```

4) Perform a static code analysis using pylint with CLIMADA's configuration `.pylintrc` (in the climada root directory). Jenkins executes it after every push.
   To do it locally, your IDE probably provides a tool, or you can run `make lint` and see the output in `pylint.log`.

5) Push to GitHub. If you're pushing this branch for the first time, use

```
git push -u origin feature/meteorite
```

   and if you're updating a branch that's already on GitHub:

```
git push
```

6) Check all the tests pass on the WCR Jenkins server (https://ied-wcr-jenkins.ethz.ch). See Emanuel's presentation for how to do this! You should regularly be pushing your code and checking this!

7) Create the pull request!

   - On the CLIMADA GitHub page, navigate to your feature branch (there's a drop-down menu above the file structure, pointing by default to `main`).
   - Above the file structure is a branch summary and an icon to the right labelled "Pull request".
   - Choose which branch you want to merge with. This will usually be `develop`, but may be another feature branch for more complex feature development.
   - Give your pull request an informative title (like a commit message).
   - Write a description of the pull request. This can usually be adapted from your branch's commit messages (you wrote informative commit messages, didn't you?), and should give a high-level summary of the changes, specific points you want the reviewers' input on, and explanations for decisions you've made. The code documentation (and any references) should cover the more detailed stuff.
   - Assign reviewers in the page's right hand sidebar. Tag anyone who might be interested in reading the code. You should already have found one or two people who are happy to read the whole request and sign it off (they could also be added to 'Assignees').
   - Create the pull request.
   - Contact the reviewers to let them know the request is live. GitHub's settings mean that they may not be alerted automatically. Maybe also let people know on the WCR Slack!

8) Talk with your reviewers

   - Use the comment/chat functionality within GitHub's pull requests - it's useful to have an archive of discussions and the decisions made.
   - Take comments and suggestions on board, but you don't need to agree with everything and you don't need to implement everything.
   - If you feel someone is asking for too many changes, prioritise, especially if you don't have time for complex rewrites.

- If the suggested changes and or features don't block functionality and you don't have time to fix them, they can be moved to Issues.
- Chase people up if they're slow. People are slow.

9) Once you implement the requested changes, respond to the comments with the corresponding commit implementing each requested change.
10) If the review takes a while, remember to merge `develop` back into the feature branch every now and again (and check the tests are still passing on Jenkins).
    Anything pushed to the branch is added to the pull request.
11) Once everyone reviewing has said they're satisfied with the code you can merge the pull request using the GitHub interface.
    Delete the branch once it's merged, there's no reason to keep it. (Also try not to re-use that branch name later.)
12) Update the `develop` branch on your local machine.

Also see the ***Reviewer Guide*** and ***Reviewer Checklist***!

3.6.6 General tips and tricks

Follow the *python do's and don't* and *performance* guides. Write small readable methods, classes and functions.

Ask for help with Git

- Git isn't intuitive, and rewinding or resetting is always work. If you're not certain what you're doing, or if you think you've messed up, send someone a message. See also our instructions for *Development with Git*.

Don't push or commit to develop or main

- Almost all new additions to CLIMADA should be merged into the `develop` branch with a pull request.
- You won't merge into the `main` branch, except for emergency hotfixes (which should be communicated to the team).
- You won't merge into the `develop` branch without a pull request, except for small documentation updates and typos.
- The above points mean you should never need to push the `main` or `develop` branches.

So if you find yourself on the `main` or `develop` branches typing `git merge ...` or `git push` stop and think again - you should probably be making a pull request.

This can be difficult to undo, so contact someone on the team if you're unsure!

Commit more often than you think, and use informative commit messages

- Committing often makes mistakes less scary to undo

```
git reset --hard HEAD
```

- Detailed commit messages make writing pull requests really easy
- Yes it's boring, but *trust me*, everyone (usually your future self) will love you when they're rooting through the git history to try and understand why something was changed

Commit message syntax guidelines

Basic syntax guidelines taken from here https://chris.beams.io/posts/git-commit/ (on 17.06.2020)

- Limit the subject line to 50 characters
- Capitalize the subject line
- Do not end the subject line with a period
- Use the imperative mood in the subject line (e.g. "Add new tests")
- Wrap the body at 72 characters (most editors will do this automatically)
- Use the body to explain what and why vs. how
- Separate the subject from body with a blank line (This is best done with a GUI. With the command line you have to use text editor, you cannot do it directly with the git command)
- Put the name of the function/class/module/file that was edited
- When fixing an issue, add the reference gh-ISSUENUMBER to the commit message e.g. "fixes gh-40." or "Closes gh-40." For more infos see here https://docs.github.com/en/enterprise/2.16/user/github/managing-your-work-on-github/closing-issues-using-keywords#about-issue-references.

What not to commit

There are a lot of things that don't belong in the Git repository:

- Don't commit data, except for config files and very small files for tests.
- Don't commit anything containing passwords or authentication credentials or tokens. (These are annoying to remove from the Git history.) Contact the team if you need to manage authorisations within the code.
- Don't commit anything that can be created by the CLIMADA code itself

If files like this are going to be present for other users as well, add them to the repository's `.gitignore`.

Jupyter Notebook metadata

Git compares file versions by text tokens. Jupyter Notebooks typically contain a lot of metadata, along with binary data like image files. Simply re-running a notebook can change this metadata, which will be reported as file changes by Git. This causes excessive Diff reports that cannot be reviewed conveniently.

To avoid committing changes of unrelated metadata, open Jupyter Notebooks in a text editor instead of your browser renderer. When committing changes, make sure that you indeed only commit things you *did* change, and revert any changes to metadata that are not related to your code updates.

Several code editors use plugins to render Jupyter Notebooks. Here we collect the instructions to inspect Jupyter Notebooks as plain text when using them:

- **VSCode**: Open the Jupyter Notebook. Then open the internal command prompt (`Ctrl` + `Shift` + `P` or `Cmd` + `Shift` + `P` on macOS) and type/select 'View: Reopen Editor with Text Editor'

Log ideas and bugs as GitHub Issues

If there's a change you might want to see in the code - something that generalises, something that's not quite right, or a cool new feature - it can be set up as a GitHub Issue. Issues are pages for conversations about changes to the codebase and for logging bugs, and act as a 'backlog' for the CLIMADA project.

For a bug, or a question about functionality, make a minimal working example, state which version of CLIMADA you are using, and post it with the Issue.

How not to mess up the timeline

Git builds the repository through incremental edits. This means it's great at keeping track of its history. But there are a few commands that *edit* this history, and if histories get out of sync on different copies of the repository you're going to have a bad time.

- Don't rebase any commits that already exist remotely!
- Don't `--force` anything that exists remotely unless you know what you're doing!
- Otherwise, you're unlikely to do anything irreversible
- You can do what you like with commits that only exist on your machine.

That said, doing an interactive rebase to tidy up your commit history *before* you push it to GitHub is a nice friendly gesture :)

Do not fast forward merges

(This shouldn't be relevant - all your merges into `develop` should be through pull requests, which doesn't fast forward. But:)

Don't fast forward your merges unless your branch is a single commit. Use `git merge --no-ff ...`

The exceptions is when you're merging `develop` into your feature branch.

Merge the remote develop branch into your feature branch every now and again

- This way you'll find conflicts early

```
git checkout develop
git pull
git checkout feature/myfeature
git merge develop
```

Create frequent pull requests

I said this already:

- It structures your workflow
- It's easier for reviewers
- If you're going to break something for other people you all know sooner
- It saves work for the rest of the team right before a release

Whenever you do something with CLIMADA, make a new local branch

You never know when a quick experiment will become something you want to save for later.

But do not do everything in the CLIMADA repository

- If you're running CLIMADA rather than developing it, create a new folder, initialise a new repository with `git init` and store your scripts and data there
- If you're writing an extension to CLIMADA that doesn't change the model core, create a new folder, initialise a new repository with `git init` and import CLIMADA. You can always add it to the model later if you need to.

Questions

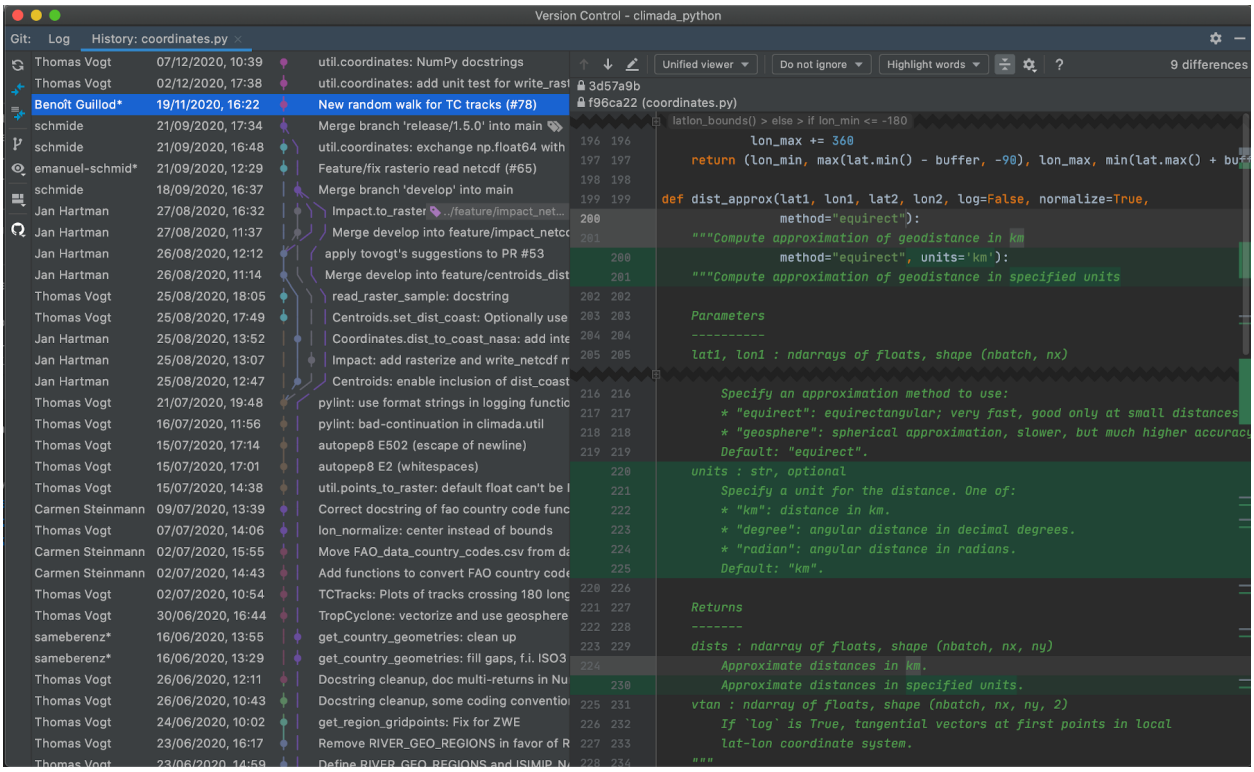
https://xkcd.com/1597/

3.7 Development with Git

Here we provide a detailed instruction to the use of Git and GitHub and their workflows, which are essential to the code development of CLIMADA.

3.7.1 Git and GitHub

- Git's not that scary
  - 95% of your work on Git will be done with the same handful of commands (the other 5% will always be done with careful Googling)
  - Almost everything in Git can be undone by design (but use `rebase`, `--force` and `--hard` with care!)
  - Your favourite IDE (Spyder, PyCharm, …) will have a GUI for working with Git, or you can download a standalone one.
- The Git Book is a great introduction to how Git works and to using it on the command line.
- Consider using a GUI program such as "git desktop" or "Gitkraken" to have a visual git interface, in particular at the beginning. Your python IDE is also likely to have a visual git interface.
- Feel free to ask for help



What we assume you know

We're assuming you're all familiar with the basics of Git.

- What (and why) is version control
- How to clone a repository
- How to make a commit and push it to GitHub
- What a branch is, and how to make one
- How to merge two branches
- The basics of the GitHub website

If you're not feeling great about this, we recommend

- sending me a message so we can arrange an introduction with CLIMADA
- exploring the Git Book

Terms we'll be using today

These are terms that will come up a lot, so let's make sure we know them

- local versus remote
  - Our **remote** repository is hosted on GitHub. This is the central location where all updates to CLIMADA that we want to share end up. If you're updating CLIMADA for the community, your code will end up here too.
  - Your **local** repository is the copy you have on the machine you're working on, and where you do your work.
  - Git calls the (first, default) remote the `origin`

  - (It's possible to set more than one remote repository, e.g. you might set one up on a network-restricted computing cluster)
- push, pull and pull request
  - You **push** your work when you send it from your local machine to the remote repository
  - You **pull** from the remote repository to update the code on your local machine
  - A **pull request** is a standardised review process on GitHub. Usually it ends with one branch merging into another
- Conflict resolution
  - Sometimes two people have made changes to the same bit of code. Usually this comes up when you're trying to merge branches. The changes have to be manually compared and the code edited to make sure the 'correct' version of the code is kept.

3.7.2 Gitflow

Gitflow is a particular way of using git to organise projects that have

- multiple developers
- working on different features
- with a release cycle

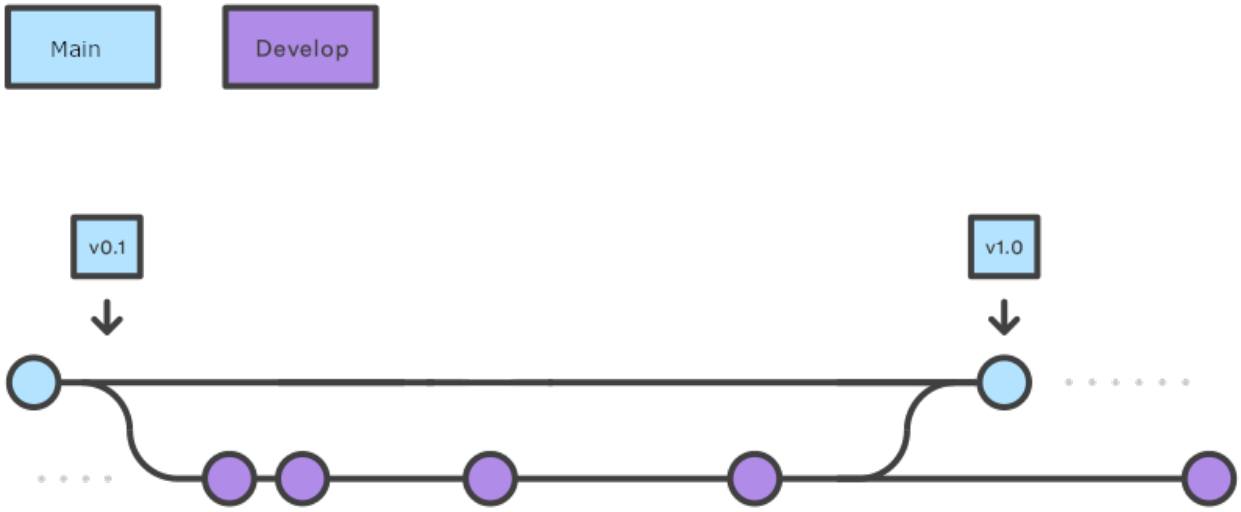
It means that

- there's always a stable version of the code available to the public
- the chances of two developers' code conflicting are reduced
- the process of adding and reviewing features and fixes is more standardised for everyone

Gitflow is a *convention*, so you don't need any additional software.

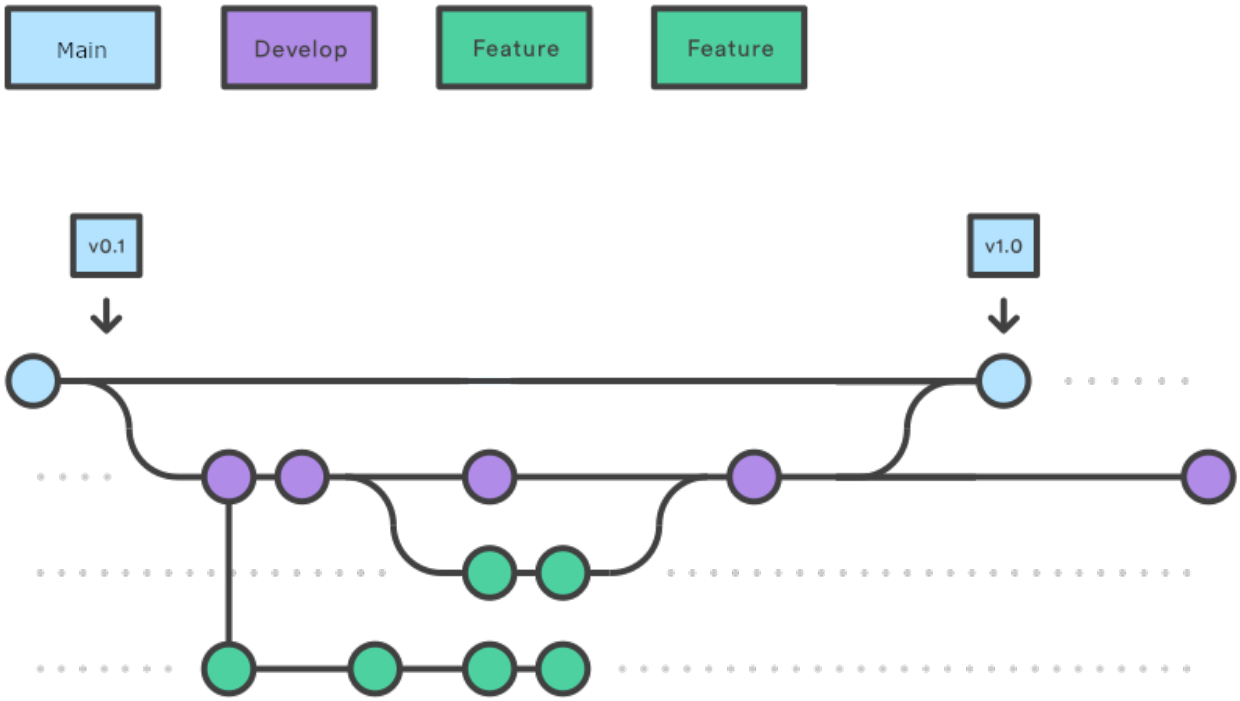
- … but if you want you can get some: a popular extension to the git command line tool allows you to issue more intuitive commands for a Gitflow workflow.
- Mac/Linux users can install git-flow from their package manager, and it's included with Git for Windows

Gitflow works on the `develop` branch instead of `main`



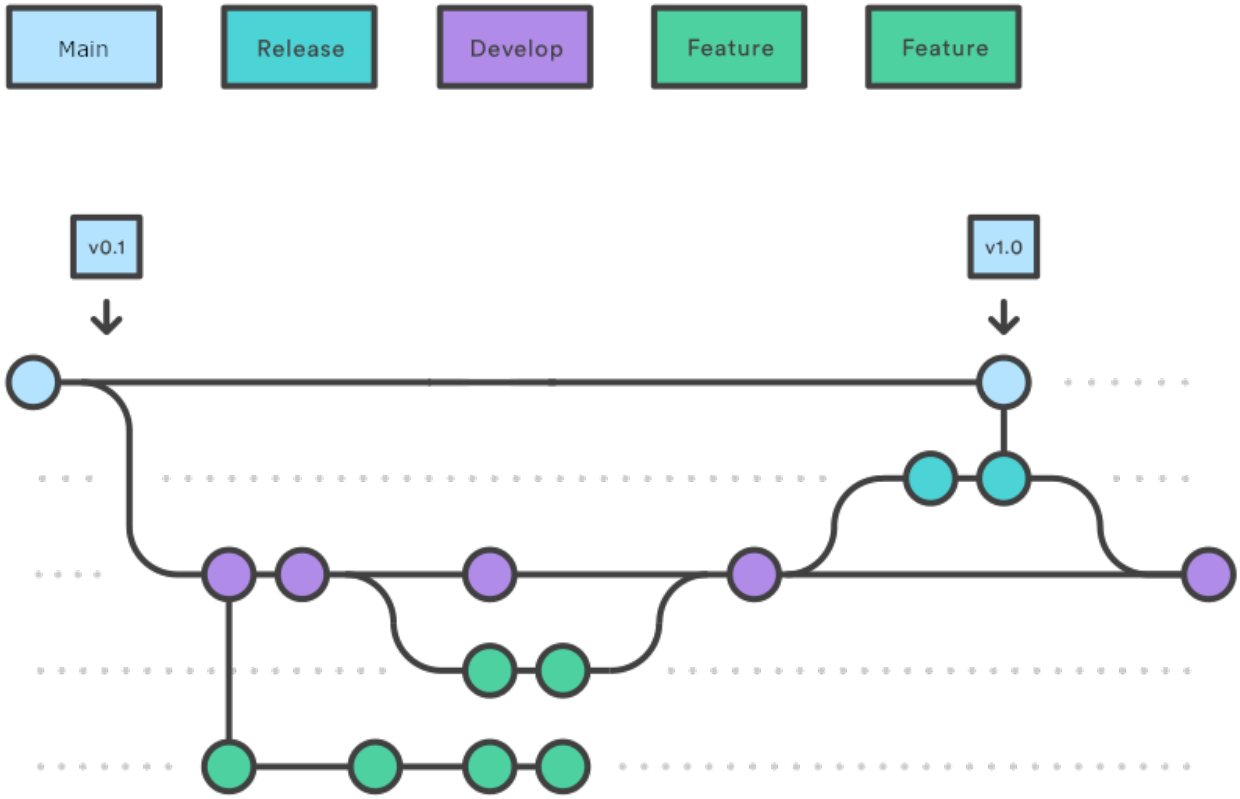
- The critical difference between Gitflow and 'standard' git is that almost all of your work takes place on the `develop` branch, instead of the `main` (formerly `master`) branch.
- The `main` branch is reserved for planned, stable product releases, and it's what the general public download when they install CLIMADA. The developers almost never interact with it.

Gitflow is a feature-based workflow



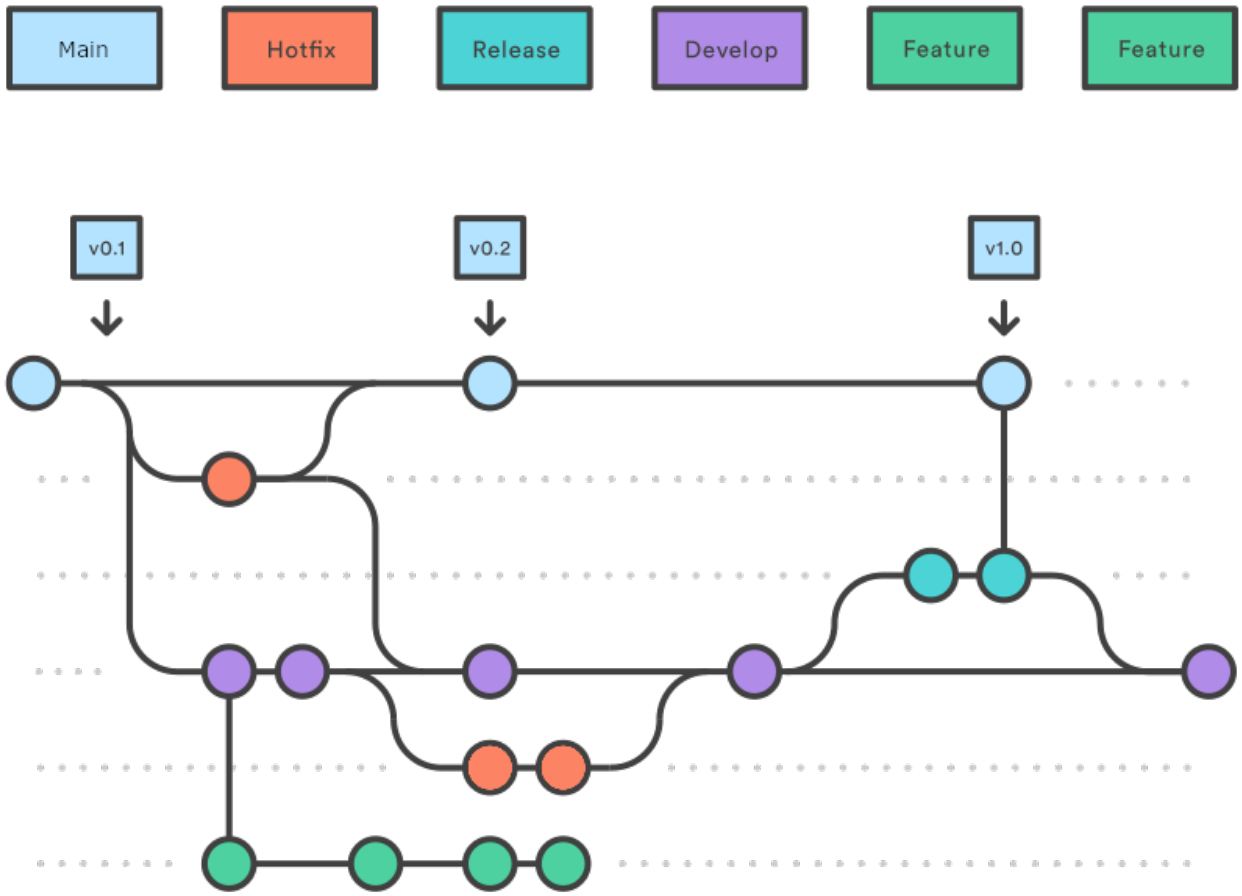
- This is common to many workflows: when you want to add something new to the model you start a new branch, work on it locally, and then merge it back into `develop` **with a pull request** (which we'll cover later).
- By convention we name all CLIMADA feature branches `feature/*` (e.g. `feature/meteorite`).
- Features can be anything, from entire hazard modules to a smarter way to do one line of a calculation. Most of the work you'll do on CLIMADA will be a features of one size or another.
- We'll talk more about developing CLIMADA features later!

Gitflow enables a regular release cycle



- A release is usually more complex than merging `develop` into `main`.
- So for this a `release-*` branch is created from `develop`. We'll all be notified repeatedly when the deadline is to submit (and then to review) pull requests so that you can be included in a release.
- The core developer team (mostly Emanuel) will then make sure tests, bugfixes, documentation and compatibility requirements are met, merging any fixes back into `develop`.
- On release day, the release branch is merged into `main`, the commit is tagged as a release and the release notes are published on the GitHub at https://github.com/CLIMADA-project/climada_python/releases

Everything else is hotfixes



- The other type of branch you'll create is a hotfix.
- Hotfixes are generally small changes to code that do one thing, fixing typos, small bugs, or updating docstrings. They're done in much the same way as features, and are usually merged with a pull request.
- The difference between features and hotfixes is fuzzy and you don't need to worry about getting it right.
- Hotfixes will occasionally be used to fix bugs on the `main` branch, in which case they will merge into both `main` and `develop`.
- Some hotfixes are so simple - e.g. fixing a typo or a docstring - that they don't need a pull request. Use your judgement, but as a rule, if you change what the code does, or how, you should be merging with a pull request.

3.8 Continuous Integration and GitHub Actions

3.8.1 Automated Tests

On Jenkins tests are executed and analyzed automatically, in an unbiased environment. The results are stored and can be compared with previous test runs.
Jenkins has a GUI for monitoring individual tests, full test runs and test result trends.
Developers are requested to watch it. At first when they push commits to the code repository, but also later on, when other changes in data or sources may make it necessary to review and refactor code that once passed all tests. The CLIMADA Jenkins server used for continuous integration is at (https://ied-wcr-jenkins.ethz.ch) .

Developer guidelines:

- All tests must pass before submitting a pull request.
- Integration tests don't run on feature branches in Jenkins, therefore developers are requested to run them locally.
- After a pull request was accepted and the changes are merged to the develop branch, integration tests may still fail there and have to be addressed.

3.8.2 Test Coverage

Jenkins also has an interface for exploring code coverage analysis result.
This shows which part of the code has never been run in any test, by module, by function/method and even by single line of code.

Ultimately every single line of code should be tested.

Jenkins Coverage Reports

To inspect the coverage reports, check out the overview of branch builds on Jenkins. Select the branch or pull request you are interested in. Then, select "Coverage Report" in the menu on the right. Note that this menu entry might not be available if no build of that particular branch/PR succeeded.

You will see a report for every directory and file in CLIMADA. Clicking on a specific file opens a view of the file where the coverage is highlighted.

GitHub Coverage Reports

To inspect the coverage reports for the GitHub Actions (see below), click on the "Checks" tag in a pull request and then on "GitHub CI" on the left. In the summary of all tasks you will find the "Artifacts" with coverage reports provided as ZIP files. You can download these files, unzip them, and open the resulting HTML files in your browser.

Developer guidelines:

- Make sure the coverage of novel code is at 100% before submitting a pull request.

Be aware that having a code coverage alone does not grant that all required tests have been written!
The following artificial example would have a 100% coverage and still obviously misses a test for `y(False)`

```
def x(b: bool):
    if b:
        print("been here")
        return 4
    else:
        print("been there")
        return 0


def y(b: bool):
    print("been everywhere")
    return 1 / x(b)


import unittest


class TestXY(unittest.TestCase):
```

(continues on next page)

(continued from previous page)

```
    def test_x(self):
        self.assertEqual(x(True), 4)
        self.assertEqual(x(False), 0)

    def test_y(self):
        self.assertEqual(y(True), 0.25)


unittest.TextTestRunner().run(unittest.TestLoader().loadTestsFromTestCase(TestXY));
```

```
..
```

```
been here
been there
been everywhere
been here
```

```
----------------------------------------------------------------------
Ran 2 tests in 0.003s

OK
```

3.8.3 Static Code Analysis

At last Jenkins provides an elaborate GUI for pylint findings which is especially useful when working in feature branches.

Observe it!

Developer guidelines:

- *High Priority Warnings* are as severe as test failures and must be addressed at once.
- Do not introduce new *Medium Priority Warnings*.
- Try to avoid introducing *Low Priority Warnings*, in any case their total number should not increase.

3.8.4 Jenkins Projects Overview

climada_install_env

Branch: **develop**
Runs every day at 1:30AM CET

- creates conda environment from scratch
- runs core functionality system test (`make install_test`)

climada_ci_night

Branch: **develop**
Runs when climada_install_env has finished successfully

- runs all test modules
- runs static code analysis

climada_branches

Branch: **any**
Runs when a commit is pushed to the repository

- runs all test modules *outside of climada.test*
- runs static code analysis

climada_data_api

Branch: **develop**
Runs every day at 0:20AM CET

- tests availability of external data APIs

climada_data_api

Branch: **develop**
No automated running

- tests executability of CLIMADA tutorial notebooks.

3.8.5 GitHub Actions

CLIMADA has been using a private Jenkins instance for automated testing (Continuous Integration, CI). We recently adopted GitHub Actions for automated unit testing. GitHub Actions is a service provided by GitHub, which lets you configure CI/CD pipelines based on YAML configuration files. GitHub provides servers which ample computational resources to create software environments, install software, test it, and deploy it. See the GitHub Actions Overview for a technical introduction, and the Workflow Syntax for a reference of the pipeline definitions.

The CI results for each pull request can be inspected in the "Checks" tab. For GitHub Actions, users can inspect the logs of every step for every job.

Note

As of CLIMADA v4.0, the default CI technology remains Jenkins. GitHub Actions CI is currently considered experimental for CLIMADA development.

Unit Testing Guideline

This pipeline is defined by the `.github/workflows/ci.yml` file. It contains a single job which will create a CLIMADA environment with Mamba for multiple Python versions, install CLIMADA, run the unit tests, and report the test coverage as well as the simplified test results. The job has a strategy which runs it for multiple times for different Python versions. This way, we make sure that CLIMADA is compatible with all currently supported versions of Python.

The coverage reports in HTML format will be uploaded as job artifacts and can be downloaded as ZIP files. The test results are simple testing summaries that will appear as individual checks/jobs after the respective job completed.

3.9 Coding in python

3.9.1 Coding in Python: Dos and Don'ts

To Code or Not to Code?

Before you start implementing functions which then go into the climada code base, you have to ask yourself a few questions:

Has something similar already been implemented? This is far from trivial to answer! First, search for functions in the same module where you'd be implementing the new piece of code. Then, search in the `util` folders, there's a lot of functions in some of the scripts! You could also search the index (a list of all functions and global constants) in the climada documentation for key-words that may be indicative of the functionality you're looking for.

Don't expect this process to be fast!

Even if you want to implement *just* a small helper function, which might take 10mins to write, it may take you 30mins to check the existing code base! That's part of the game! Even if you found something, most likely, it's not the *exact* same thing which you had in mind. Then, ask yourself how you can re-use what's there, or whether you can easily add another option to the existing method to also fit your case, and only if it's nearly impossible or highly unreadable to do so, write your own implementation.

Can my code serve others? You probably have a very specific problem in mind. Yet, think about other use-cases, where people may have a similar problem, and try to either directly account for those, or at least make it easy to configure to other cases. Providing keyword options and hard-coding as few things as possible is usually a good thing. For example, if you want to write a daily aggregation function for some time-series, consider that other people might find it useful to have a general function that can also aggregate by week, month or year.

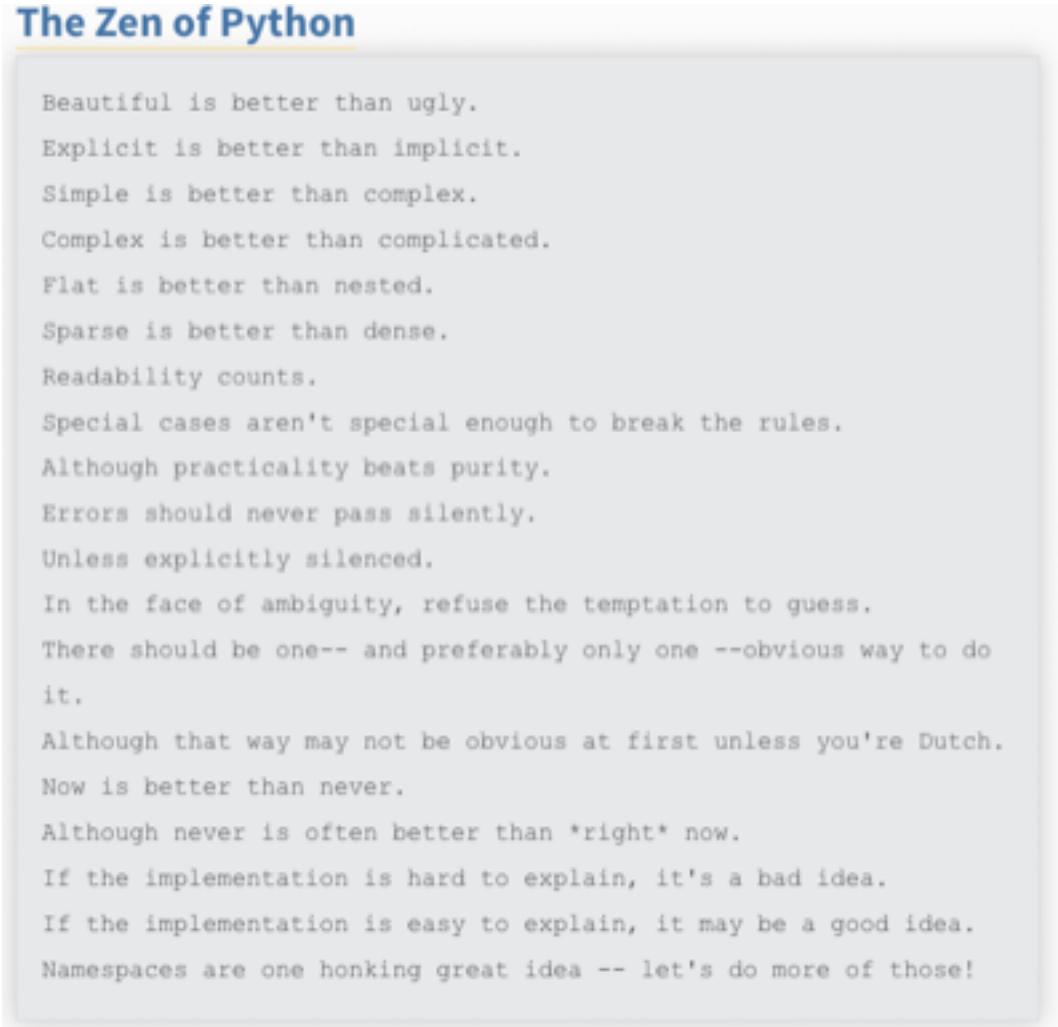
Can I get started? Before you finally start coding, be sure about placing them in a sensible location. Functions in non-util modules are actually specific for that module (e.g. a file-reader function is probably not river-flood specific, so put it into the `util` section, not the `RiverFlood` module, even if that's what you're currently working on)! If unsure, talk with other people about where your code should go.

If you're implementing more than just a function or two, or even an entirely new module, the planning process should be talked over with someone doing climada-administration.

Clean Code

A few basic principles:

- Follow the PEP 8 Style Guide. It contains, among others, recommendations on:
  - code layout
  - basic naming conventions
  - programming recommendations
  - commenting (in detail described in Chapter 4)
  - varia
- Perform a static code analysis - or: PyLint is your friend
- Follow the best practices of *Correctness - Tightness - Readability*
- Adhere to principles of pythonic coding (idiomatic coding, the "python way")



PEP 8 Quickie: Code Layout

- *Indentation*: 4 spaces per level. For continuation lines, decide between vertical alignment & hanging indentation as shown here:

```
# Vertically aligned with opening delimiter.
foo = long_function_name(var_one, var_two, var_three, var_four)


# Hanging indentation (4 additonal spaces)
def very_very_long_function_name(var_one, var_two, var_three, var_four):
    print(var_one)
```

- *Line limit*: maximum of 79 characters (docstrings & comments 72).
- *Blank lines*:
  - **Two**: Surround top-level function and class definitions;
  - **One:** Surround Method definitions inside a class
  - **Several:** may be used (sparingly) to separate groups of related functions

  - **None:** Blank lines may be omitted between a bunch of related one-liners (e.g. a set of dummy implementations).

- *Whitespaces*:
  - **None** immediately inside parentheses, brackets or braces; after trailing commas; for keyword assignments in functions.
  - **Do** for assignments (`i = i + 1`), around comparisons (>=, ==, etc.), around booleans (`and`, `or`, `not`)
  - the following 3 examples are correct:

```
spam(ham[1], {eggs: 2})
if x == 4: print x, y; x, y = y, x
def complex(real, imag=0.0):
```

- There's more in the PEP 8 guide!

PEP 8 Quickie: Basic Naming Conventions

A short typology: b (single lowercase letter); B (single uppercase letter); lowercase; lower_case_with_underscores; UPPERCASE; UPPER_CASE_WITH_UNDERSCORES; CapitalizedWords (or CapWords, or CamelCase); mixedCase; Capitalized_Words_With_Underscores (ugly!)

A few basic rules:

- packages and modules: short, all-lowercase names. Underscores can be used in the module name if it improves readability. E.g. `numpy`, `climada`
- classes: use the CapWords convention. E.g. `RiverFlood`
- functions, methods and variables: lowercase, with words separated by underscores as necessary to improve readability. E.g. `from_raster()`, `dst_meta`
- function- and method arguments: Always use `self` for the first argument to instance methods,`cls` for the first argument to class methods.
- constants: all capital letters with underscores, e.g. `DEF_VAR_EXCEL`

Use of underscores

- `_single_leading_underscore`: weak "internal use" indicator. E.g. `from M import *` does not import objects whose names start with an underscore. A side-note to this: Always decide whether a class's methods and instance variables (collectively: "attributes") should be public or non-public. If in doubt, choose non-public; it's easier to make it public later than to make a public attribute non-public. Public attributes are those that you expect unrelated clients of your class to use, with your commitment to avoid backwards incompatible changes. Non-public attributes are those that are not intended to be used by third parties; you make no guarantees that non-public attributes won't change or even be removed. Public attributes should have no leading underscores.
- `single_trailing_underscore_`: used by convention to avoid conflicts with Python keyword, e.g. `tkinter.Toplevel(master, class_='ClassName')`
- `__double_leading_and_trailing_underscore__`: "magic" objects or attributes that live in user-controlled namespaces. E.g. `__init__`, `__import__` or `__file__`. Never invent such names; only use them as documented.

There are many more naming conventions, some a bit messy. Have a look at the PEP8 style guide for more cases.

PEP 8 Quickie: Programming Recommendations

- comparisons to singletons like `None` should always be done with `is` or `is not`, never the equality operators.
- Use `is not` operator rather than `not ... is`.
- Be consistent in return statements. Either all return statements in a function should return an expression, or none of them should. Any return statements where no value is returned should explicitly state this as `return None`.

```
# Correct
def foo(x):
    if x >= 0:
        return math.sqrt(x)
    else:
        return None


# Wrong
def foo(x):
    if x >= 0:
        return math.sqrt(x)
```

- Object type comparisons should always use isinstance() instead of comparing types directly:

```
# Correct:
if isinstance(obj, int):
# Wrong:
if type(obj) is type(1)
```

- Remember: sequences (strings, lists, tuples) are false if empty; this can be used:

```
# Correct:
if not seq:
if seq:
# Wrong:
if len(seq):
if not len(seq)
```

- Don't compare boolean values to True or False using ==:

```
# Correct:
if greeting:
# Wrong:
if greeting == True:
```

- Use ".startswith() and ".endswith() instead of string slicing to check for prefixes or suffixes.

```
# Correct:
if foo.startswith('bar'):
# Wrong:
if foo[:3] == 'bar':
```

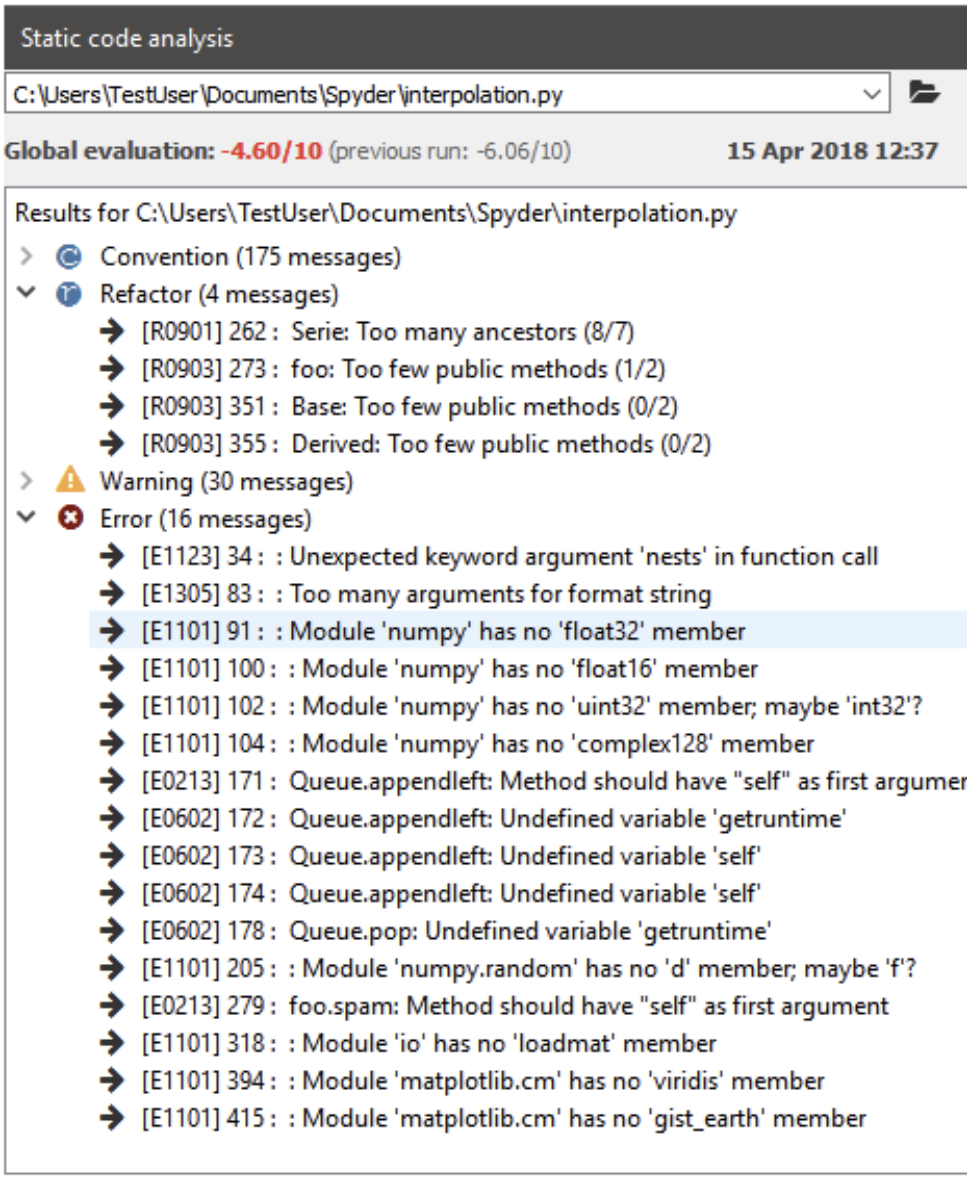
- Context managers exist and can be useful (mainly for opening and closing files

Static Code Analysis and PyLint

Static code analysis detects style issues, bad practices, potential bugs, and other quality problems in your code, all without having to actually execute it. In Spyder, this is powered by the best in class Pylint back-end, which can intelligently detect an enormous and customizable range of problem signatures. It follows the style recommended by PEP 8 and also includes the following features: Checking the length of each line, checking that variable names are well-formed according to the project's coding standard, checking that declared interfaces are truly implemented.

A detailed instruction can be found here.

In brief: In the editor, select the Code Analysis pane (if not visible, go to View -> Panes -> Code Analysis) and the file you want to be analyzed; hit the Analyze button.



The output will look somewhat similar to that:

There are 4 categories in the analysis output:

- *convention*,
- *refactor*,

- *warning*,
- *error*
- a global score regarding code quality.

All messages have a line reference and a short description on the issue. Errors *must* be fixed, as this is a no-go for actually executing the script. Warnings and refactoring messages should be taken seriously; so should be the convention messages, even though some of the naming conventions etc. may not fit the project style. This is configurable.

In general, there should be no errors and warnings left, and the overall code quality should be in the "green" range (somewhere above 5 or so).

There are advanced options to configure the type of warnings and other settings in pylint.

A few more best practices

Correctness

Methods and functions must return correct and verifiable results, not only under the best circumstances but in any possible context. I.e. ideally there should be unit tests exploring the full space of parameters, configuration and data states. This is often clearly a non-achievable goal, but still - we aim at it.

Tightness

- Avoid code redundancy.
- Make the program efficient, use profiling tools for detection of bottlenecks.
- Try to minimize memory consumption.
- Don't introduce new dependencies (library imports) when the desired functionality is already covered by existing dependencies.
- Stick to already supported file types.

Readability

- Write complete Python Docstrings.
- Use meaningful method and parameter names, and always annotate the data types of parameters and return values.
- No context-dependent return types! Also: Avoid `None` as return type, rather raise an Exception instead.
- Be generous with defining Exception classes.
- Comment! Comments are welcome to be redundant. And whenever there is a particular reason for the way something is done, comment on it! See below for more detail.
- For functions which implement mathematical/scientific concepts, add the actual mathematical formula as comment or to the Doctstrings. This will help maintain a high level of scientific accuracy. E.g. How is are the random walk tracks computed for tropical cyclones?

Pythonic Code

In Python, there are certain structures that are specific to the language, or at least the syntax of how to use them. This is usually referred to as "pythonic" code.

There is an extensive overview on on crucial "pythonic" structures and methods in the Python 101 library.

A few important examples are:

- iterables such as dictionaries, tuples, lists

- iterators and generators (a very useful construct when it comes to code performance, as the implementation of generators avoids reading into memory huge iterables at once, and allows to read them lazily on-the-go; see this blog post for more details)
- f-strings ("formatted string literals," have an f at the beginning and curly braces containing expressions that will be replaced with their values:

```
>>> name = "Eric"
>>> age = 74
>>> f"Hello, {name}. You are {age}."
'Hello, Eric. You are 74.'
```

- decorators (a design pattern in Python that allows a user to add new functionality to an existing object without modifying its structure). Something like:

```
@uppercase_decorator
def say_hi():
    return "hello there"
```

- type checking (Python is a dynamically typed language; also: cf. "Duck typing". Yet, as a best practice, variables should not change type once assigned)
- Do not use mutable default arguments in your functions (e.g. lists). For example, if you define a function as such:

```
def function(x, list=[]):
    default_list.append(x)
```

  Your list will be mutated for future calls of the functions too. The correct implementation would be the following:

```
def func(x, list=None):
 list = [] if list is None
```

- lambda functions (little, anonymous functions, sth like `high_ord_func(2, lambda x: x * x)`)
- list comprehensions (a short and possibly elegant syntax to create a new list in one line, sth like `newlist = [x for x in range(10) if x < 5]` returns `[0, 1, 2, 3, 4]`)

It is recommended to look up the above concepts in case not familiar with them.

Commenting & Documenting

What is what

Comments are for developers. They describe parts of the code where necessary to facilitate the understanding of programmers. They are marked by putting a `#` in front of every comment line (for multi-liners, wrapping them inside triple double quotes `"""` is basically possible, but discouraged to not mess up with docstrings). A *documentation string (docstring)* is a string that describes a module, function, class, or method definition. The docstring is a special attribute of the object (`object.__doc__`) and, for consistency, is surrounded by triple double quotes (`"""`). This is also where elaboration of the scientific foundation (explanation of used formulae, etc.) should be documented.

A few general rules:

- Have a look at this blog-post on commenting basics
- Comments should be D.R.Y ("Don't Repeat Yourself.")

- Obvious naming conventions can avoid unnecessary comments (cf. `families_by_city[city]` vs. `my_dict[p]`)
- comments should rarely be longer than the code they support
- All public methods need a doc-string. See below for details on the convention used within the climada project.
- Non-public methods that are not immediately obvious to the reader should at least have a short comment after the def line:

```
def complicated_function(s):
    # This function does something complicated
```

Numpy-style docstrings

Full reference can be found here. The standards are such that they use re-structured text (reST) syntax and are rendered using Sphinx.

There are several sections in a docstring, with headings underlined by hyphens (`---`). The sections of a function's docstring are:

1. *Short summary*: A one-line summary that does not use variable names or the function name

```
def add(a, b):
    """The sum of two numbers.

    """
```

2. *Deprecation warning* (use if applicable): to warn users that the object is deprecated, including version the object that was deprecated, and when it will be removed, reason for deprecation, new recommended way of obtaining the same functionality. Use the `deprecated` Sphinx directive:

```
.. deprecated:: 1.6.0
          `ndobj_old` will be removed in NumPy 2.0.0, it is replaced by
          `ndobj_new` because the latter works also with array subclasses.
```

3. *Extended Summary*: A few sentences giving an extended description to clarify functionality, not to discuss implementation detail or background theory (see `Notes` section below!)

4. *Parameters*:

```
Parameters
----------
x : type
    Description of parameter `x`.
```

Description of the function arguments, keywords and their respective types. Enclose variables in single backticks in the description. The colon must be preceded by a space, or omitted if the type is absent. For the parameter types, be as precise as possible. If it is not necessary to specify a keyword argument, use `optional` after the type specification: e.g. `x: int, optional`. Default values of optional parameters can also be detailed in the description. (e.g. `... description of parameter ... (default is -1)`)

5. *Returns*: Explanation of the returned values and their types. Similar to the Parameters section, except the name of each return value is optional, type isn't. If both the name

and type are specified, the Returns section takes the same form as the Parameters section.

```
Returns
-------
err_code : int
    Non-zero value indicates error code, or zero on success.
err_msg : str or None
    Human readable error message, or None on success.
```

There is a range of other sections that can be included, if sensible and applicable, such as `Yield` (for generator functions only), `Raises` (which errors get raised and under what conditions), `See also` (refer to related code), `Notes` (additional information about the code, possibly including a discussion of the algorithm; may include mathematical equations, written in LaTeX format), `References`, `Examples`(to illustrate usage).

Importing

General remarks

- Imports should be grouped in the following order:
  - Standard library imports (such as `re`, `math`, `datetime`, cf. here)
  - Related third party imports (such as `numpy`)
  - Local application/library specific imports (such as `climada.hazard.base`)
- You should put a blank line between each group of imports.
- Don't introduce new dependencies (library imports) when the desired functionality is already covered by existing dependencies.

Avoid circular importing!!

Circular imports are a form of circular dependencies that are created with the import statement in Python; e.g. module A loads a method in module B, which in turn requires loading module A. This can generate problems such as tight coupling between modules, reduced code reusability, more difficult maintenance. Circular dependencies can be the source of potential failures, such as infinite recursions, memory leaks, and cascade effects. Generally, they can be resolved with better code design. Have a look here for tips to identify and resolve such imports.

Varia

- there are absolute imports (uses the full path starting from the project's root folder) and relative imports (uses the path starting from the current module to the desired module; usually in the for `from .<module/package> import X`; dots `.` indicate how many directories upwards to traverse. A single dot corresponds to the current directory; two dots indicate one folder up; etc.)
- generally try to avoid star imports (e.g. `from packagename import *`)

Importing utility functions

When importing CLIMADA utility functions (from climada.util), the convention is to import the function as "u_name_of_function", e.g.:

```
from climada.util import coordinates as u_coord
u_coord.make_map()
```

How to structure a method or function

To clarify ahead: The questions of how to structure an entire module, or even "just" a class, are not treated here. For this, please get in contact with the repository admins to help you go devise a plan.

The following few principles should be adhered to when designing a function or method (which is simply the term for a function inside a class):

- have a look at this blog-post summarizing a few important points to define your function (key-words *abstraction, reusability, modularity*)
- separate algorithmic computations and data curation
- adhere to a maximum method length (rule of thumb: if it doesn't fit your screen, it's probably an indicator that you should refactor into sub-functions)
- divide functions into single purpose pieces (one function, one goal)

Debugging

When writing code, you will encounter bugs and hence go through (more or less painful) debugging. Depending on the IDE you use, there are different debugging tools that will make your life much easier. They offer functionalities such as stopping the execution of the function just before the bug occurs (via breakpoints), allowing to explore the state of defined variables at this moment of time.

For spyder specifically, have a look at the instructions on how to use ipdb

3.9.2 Exception Handling and Logging

Exception handling and logging are two important components of programming, in particular for debugging purposes. Detailed technical guides are available online (e.g., Loggin, Error and Exceptions). Here we only repeat a few key points and list a few guidelines for CLIMADA.

Exception handling

CLIMADA guidelines

1. Catch specific exceptions if possible, i.e, if not needed do not catch all exceptions.
2. Do not catch exception if you do not handle them.
3. Make a clear explanatory message when you raise an error (similarly to when you use the logger to inform the user). Think of future users and how it helps them understanding the error and debugging their code.
4. Catch an exception when it arises.
5. When you catch an exception and raise an error, it is in often (but not always) a good habit to not throw away the first caught exception as it may contain useful information for debugging. (use `raise Error from`)

```
# Bad (1)
x = 1
try:
    l = len(events)
    if l < 1:
        print("l is too short")
except:
    pass
```

```
# Still bad (2)
try:
    l = len(events)
    if l < 1:
        print("l is too short")
except TypeError:
    pass
```

```
# Better, but still unsufficient (3)
try:
    l = len(events)
    if l < 1:
        raise ValueError("To compute an impact there must be at least one event.")
except TypeError:
    raise TypeError("The provided variable events is not a list")
```

```
# Even better (4)
try:
    l = len(events)
except TypeError:
    raise TypeError("The provided variable events is not a list")
if l < 1:
    raise ValueError("To compute an impact there must be at least one event.")
```

```
# Even better (5)
try:
    l = len(events)
except TypeError as tper:
    raise TypeError("The provided variable events is not a list") from tper
if l < 1:
    raise ValueError("To compute an impact there must be at least one event.")
```

Exceptions reminder

Why do we bother to handle exceptions?

- The most essential benefit is to inform the user of the error, while still allowing the program to proceed.

Logging

CLIMADA guidelines

- In CLIMADA, you cannot use printing. Any output must go into the LOGGER.
- For any logging messages, always think about the audience. What would a user or developer need for information? This also implies to carefully think about the correct LOGGER level. For instance, some information is for debugging, then use the debug level. In this case, make sure that the message actually helps the debugging process! Some message might just inform the user about certain default parameters, then use the inform level. See below for more details about logger levels.
- Do not overuse the LOGGER. Think about which level of logging. Logging errors must be useful for debugging.

You can set the level of the LOGGER using `climada.util.config.LOGGER.setLevel(logging.XXX)`. This way you can for instance 'turn-off' info messages when you are making an application. For example, setting the logger to the "ERROR" level, use:

```
import logging
from climada.util.config import LOGGER

LOGGER.setLevel(logging.ERROR)
```

What levels to use in CLIMADA?

- Debug: what you would print while developing/debugging
- Info: information for example in the check instance
- Warning: whenever CLIMADA fills in values, makes an extrapolation, computes something that might potentially lead to unwanted results (e.g., the 250year damages extrapolated from data over 20 years)

No known use case:

- Error: instead, raise an Error and add the message (raise ValueError("Error message"))
- Critical: …

Reminder about Logging

"Logging is a means of tracking events that happen when some software runs."

When to use logging

"Logging provides a set of convenience functions for simple logging usage. These are debug(), info(), warning(), error() and critical(). To determine when to use logging, see the table below, which states, for each of a set of common tasks, the best tool to use for it."

Task you want to perform	The best tool for the task
Display console output for ordinary usage of a command line script or program	`print()`
Report events that occur during normal operation of a program (e.g. for status monitoring or fault investigation)	`logging.info()` (or `logging.debug()` for very detailed output for diagnostic purposes)
Issue a warning regarding a particular runtime event	`warnings.warn()` in library code if the issue is avoidable and the client application should be modified to eliminate the warning `logging.warning()` if there is nothing the client application can do about the situation, but the event should still be noted
Report an error regarding a particular runtime event	Raise an exception
Report suppression of an error without raising an exception (e.g. error handler in a long-running server process)	`logging.error()`, `logging.exception()` or `logging.critical()` as appropriate for the specific error and application domain

Logger level

"The logging functions are named after the level or severity of the events they are used to track. The standard levels and their applicability are described below (in increasing order of severity):"

Level	When it's used
`DEBUG`	Detailed information, typically of interest only when diagnosing problems.
`INFO`	Confirmation that things are working as expected.
`WARNING`	An indication that something unexpected happened, or indicative of some problem in the near future (e.g. 'disk space low'). The software is still working as expected.
`ERROR`	Due to a more serious problem, the software has not been able to perform some function.
`CRITICAL`	A serious error, indicating that the program itself may be unable to continue running.

3.9.3 Python performance tips and best practice for CLIMADA developers

This guide covers the following recommendations:

 Use profiling tools to find and assess performance bottlenecks. **Replace for-loops** by built-in functions and efficient external implementations. **Consider algorithmic performance**, not only implementation performance. **Get familiar with NumPy:** vectorized functions, slicing, masks and broadcasting. • **Miscellaneous:** sparse arrays, Numba, parallelization, huge files (xarray), memory, pickle format. ⚠ **Don't over-optimize** at the expense of readability and usability.

Profiling

Python comes with powerful packages for the **performance assessment** of your code. Within IPython and notebooks, there are several magic commands for this task:

- `%time`: Time the execution of a single statement
- `%timeit`: Time repeated execution of a single statement for more accuracy
- `%%timeit` Does the same as `%timeit` for a whole cell
- `%prun`: Run code with the profiler
- `%lprun`: Run code with the line-by-line profiler
- `%memit`: Measure the memory use of a single statement
- `%mprun`: Run code with the line-by-line memory profiler

More information on profiling in the Python Data Science Handbook.

Also useful: unofficial Jupyter extension Execute Time.

While it's easy to assess how fast or slow parts of your code are, including finding the bottlenecks, **generating an improved version of it is much harder**. This guide is about **simple best practices** that everyone should know who works with Python, especially when models are performance-critical.

In the following, we will **focus on arithmetic operations** because they play an important role in CLIMADA. Operations on non-numeric objects like strings, graphs, databases, file or network IO might be just as relevant inside and outside of the CLIMADA context. Some of the tips presented here do also apply to other contexts, but **it's always worth looking for context-specific performance guides**.

General considerations

This section will be concerned with:

- **for-loops** and built-ins
- **external implementations** and converting data structures
- **algorithmic efficiency**
- **memory usage**

Σ As this section's toy example, let's assume we want to sum up all the numbers in a list:

```
list_of_numbers = list(range(10000))
```

for-loops

A developer with a background in C++ would probably loop over the entries of the list:

```
%%timeit
result = 0
for i in list_of_numbers:
    result += i
```

```
332 µs ± 65.7 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

The built-in function `sum` is much faster:

```
%timeit sum(list_of_numbers)
```

```
54.9 µs ± 5.63 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

The timing improves by a factor of 5-6 and this is not a coincidence: **for-loops generally tend to get prohibitively expensive** when the number of iterations increases.

When you have a for-loop with many iterations in your code, check for built-in functions or efficient external implementations of your programming task.

A special case worth noting are `append` operations on lists which can often be replaced by more efficient *list comprehensions*.

Converting data structures

When you find an external library that solves your task efficiently, always consider that it might be necessary to convert your data structure which takes time.

For arithmetic operations, NumPy is a great library, but if your data comes as a Python list, NumPy will spend quite some time converting it to a NumPy array:

```
import numpy as np

%timeit np.sum(list_of_numbers)
```

```
572 µs ± 80 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

This operation is even slower than the for-loop!

However, if you can somehow obtain your data in the form of **NumPy arrays from the start**, or if you perform many operations that might compensate for the conversion time, the gain in performance can be considerable:

```
# do the conversion outside of the `%timeit`
ndarray_of_numbers = np.array(list_of_numbers)
%timeit np.sum(ndarray_of_numbers)
```

```
10.6 µs ± 1.56 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Indeed, this is 5-6 times faster than the built-in `sum` and 20-30 times faster than the for-loop.

Always consider several implementations

Even for such a basic task as summing, there exist several implementations whose performance can vary more than you might expect:

```
%timeit ndarray_of_numbers.sum()
%timeit np.einsum("i->", ndarray_of_numbers)
```

```
9.07 µs ± 1.39 µs per loop (mean ± std. dev. of 7 runs, 100000 loops each)
5.55 µs ± 383 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

This is up to 50 times faster than the for-loop. More information about the `einsum` function will be given in the NumPy section of this guide.

Efficient algorithms

Consider algorithmic performance, not only implementation performance.

All of the examples above do exactly the same thing, algorithmically. However, often the largest performance improvements can be obtained from **algorithmic changes**. This is the case when your model or your data contain symmetries or more complex structure that allows you to skip or boil down arithmetic operations.

In our example, we are summing the numbers from 1 to 10,000 and it's a well known mathematical theorem that this can be done using only two multiplications and an increment:

```
n = max(list_of_numbers)
%timeit 0.5 * n * (n + 1)
```

```
83.1 ns ± 2.5 ns per loop (mean ± std. dev. of 7 runs, 10000000 loops each)
```

Not surprisingly, This is almost 100 times faster than even the fastest implementation of the 10,000 summing operations listed above.

You don't need a degree in maths to find algorithmic improvements. Other algorithmic improvements that are often easy to detect are:

- **Filter your data set as much as possible** to perform operations only on those entries that are really relevant. ***Example:*** When computing a physical hazard (e.g. extreme wind) with CLIMADA, restrict to Centroids on land unless you know that some of your exposure is off shore.
- Make sure to **detect inconsistent or trivial input parameters early on**, before starting any operations. ***Example:*** If your code does some complicated stuff and applies a user-provided normalization factor at the very end, make sure to check that the factor is not 0 before you start applying those complicated operations.

In general: Before starting to code, take pen and paper and write down what you want to do from an algorithmic perspective.

Memory usage

Be careful with deep copies of large data sets and only load portions of large files into memory as needed.

Write your code in such a way that you **handle large amounts of data chunk by chunk** so that Python does not need to load everything into memory before performing any operations. When you do, Python's generators might help you with the implementation.

Allocating unnecessary amounts of memory might slow down your code substantially due to swapping.

NumPy-related tips and best practice

As mentioned above, arithmetic operations in Python can profit a lot from NumPy's capabilities. In this section, we collect some tips how to make use of NumPy's capabilities when performance is an issue.

Vectorized functions

We mentioned above that Python's **for-loops are really slow**. This is even more important when looping over the entries in a NumPy array. Fortunately, NumPy's masks, slicing notation and vectorization capabilities help to avoid for-loops in almost every possible situation:

```
# TASK: compute the column-sum of a 2-dimensional array
input_arr = np.random.rand(100, 3)
```

```
%%timeit
# SLOW: summing over columns using loops
output = np.zeros(100)
for row_i in range(input_arr.shape[0]):
    for col_i in range(input_arr.shape[1]):
        output[row_i] += input_arr[row_i, col_i]
```

```
145 µs ± 5.47 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
```

```
# FASTER: using NumPy's vectorized `sum` function with `axis` attribute
%timeit output = input_arr.sum(axis=1)
```

```
4.23 µs ± 216 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

In the special case of multiplications and sums (linear operations) over the axes of two multi-dimensional arrays, NumPy's **`einsum`** is even faster:

```
%timeit output = np.einsum("ij->i", input_arr)
```

```
2.38 µs ± 214 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Another `einsum` example: **Euclidean norms**

```
many_vectors = np.random.rand(1000, 3)
%timeit np.sqrt((many_vectors**2).sum(axis=1))
%timeit np.linalg.norm(many_vectors, axis=1)
%timeit np.sqrt(np.einsum("...j,...j->...", many_vectors, many_vectors))
```

```
24.4 µs ± 2.18 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
26.5 µs ± 2.44 µs per loop (mean ± std. dev. of 7 runs, 10000 loops each)
9.5 µs ± 91.1 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

For more information about the capabilities of NumPy's `einsum` function, refer to the official NumPy documentation. However, note that future releases of NumPy will eventually improve the performance of core functions, so that `einsum` will become an example of over-optimization (see above) at some point. Whenever you use `einsum`, consider adding a comment that explains what it does for users that are not familiar with `einsum`'s syntax.

Not only `sum`, but many NumPy functions come with similar vectorization capabilities. You can take minima, maxima, means or standard deviations along selected axes. But did you know that the same is true for the `diff` and `argmin` functions?

```
arr = np.random.randint(low=0, high=10, size=(4, 3))
arr
```

```
array([[4, 2, 6],
       [2, 3, 4],
       [3, 3, 3],
       [3, 2, 4]])
```

```
arr.argmin(axis=1)
```

```
array([1, 0, 0, 1])
```

Broadcasting

When operations are performed on several arrays, possibly of differing shapes, be sure to use NumPy's **broadcasting** capabilities. This will save you a lot of memory and time when performing arithmetic operations.

Example: We want to multiply the columns of a two-dimensional array by values stored in a one-dimensional array. There are two naive approaches to this:

```
input_arr = np.random.rand(100, 3)
col_factors = np.random.rand(3)
```

```
# SLOW: stack/tile the one-dimensional array to be two-dimensional
%timeit output = np.tile(col_factors, (input_arr.shape[0], 1)) * input_arr
```

```
5.67 µs ± 718 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
%%timeit
# SLOW: loop over columns and factors
output = input_arr.copy()
for i, factor in enumerate(col_factors):
    output[:, i] *= factor
```

```
9.63 µs ± 95.2 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

The idea of *broadcasting* is that NumPy **automatically matches axes from right to left and implicitly repeats data along missing axes** if necessary:

```
%timeit output = col_factors * input_arr
```

```
1.41 µs ± 51.7 ns per loop (mean ± std. dev. of 7 runs, 1000000 loops each)
```

For automatic broadcasting, the *trailing* dimensions of two arrays have to match. NumPy is matching the shapes of the arrays *from right to left*. If you happen to have arrays where other dimensions match, **you have to tell NumPy which dimensions to add by adding an axis of length 1 for each missing dimension**:

```
input_arr = np.random.rand(3, 100)
row_factors = np.random.rand(3)
output = row_factors.reshape(3, 1) * input_arr
```

Because this concept is so important, there is a short-hand notation for adding an axis of length 1. In the slicing notation, **add `None` in those positions where broadcasting should take place**.

```
input_arr = np.random.rand(3, 100)
row_factors = np.random.rand(3)
output = row_factors[:, None] * input_arr
```

```
input_arr = np.random.rand(7, 3, 5, 4, 6)
factors = np.random.rand(7, 3, 4)
output = factors[:, :, None, :, None] * input_arr
```

A note on in-place operations

While **in-place operations** are generally faster than long and explicit expressions, they shouldn't be over-estimated when looking for performance bottlenecks. Often, the loss in code readability is not justified because NumPy's memory management is really fast.

Don't over-optimize!

```
shape = (1200, 1700)
arr_a = np.random.rand(*shape)
arr_b = np.random.rand(*shape)
arr_c = np.random.rand(*shape)
```

```
# long expression in one line
%timeit arr_d = arr_c * (arr_a + arr_b) - arr_a + arr_c
```

```
17.3 ms ± 820 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
%%timeit
# almost same performance: in-place operations
arr_d = arr_a + arr_b
arr_d *= arr_c
arr_d -= arr_a
arr_d += arr_c
```

```
17.4 ms ± 618 µs per loop (mean ± std. dev. of 7 runs, 10 loops each)
```

```
# You may want to install the module "memory_profiler" first: activate the␣
↪environment climada_env in an Anaconda prompt,
# type "pip install memory_profiler" and execute it
%load_ext memory_profiler
```

```
# long expression in one line
%memit arr_d = arr_c * (arr_a + arr_b) - arr_a + arr_c
```

```
peak memory: 156.68 MiB, increment: 31.20 MiB
```

```
%%memit
# almost same memory usage: in-place operations
arr_d = arr_a + arr_b
arr_d *= arr_c
arr_d -= arr_a
arr_d += arr_c
```

```
peak memory: 157.27 MiB, increment: 0.00 MiB
```

Miscellaneous

Sparse matrices

In many contexts, we deal with sparse matrices or sparse data structures, i.e. two-dimensional arrays where most of the entries are 0. In CLIMADA, this is especially the case for the `intensity` attributes of `Hazard` objects. This kind of data is usually handled using SciPy's submodule scipy.sparse.

When dealing with sparse matrices make sure that you always understand exactly which of your variables are sparse and which are dense and only switch from sparse to dense when absolutely necessary.

Multiplications (`multiply`) and matrix multiplications (`dot`) are often faster than operations that involve masks or indexing.

As an example for the last rule, consider the problem of multiplying certain rows of a sparse array by a scalar:

```
import scipy.sparse as sparse

array = np.tile(np.array([0, 0, 0, 2, 0, 0, 0, 1, 0], dtype=np.float64), (100, 80))
row_mask = np.tile(np.array([False, False, True, False, True], dtype=bool), (20,))
```

In the following cells, note that the code in the first line after the `%%timeit` statement is not timed, it's the setup line.

```
%%timeit sparse_array = sparse.csr_matrix(array)
sparse_array[row_mask, :] *= 5
```

```
/home/tovogt/.local/share/miniconda3/envs/tc/lib/python3.7/site-packages/scipy/sparse/
↪data.py:55: RuntimeWarning: overflow encountered in multiply
  self.data *= other
```

```
1.52 ms ± 155 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%%timeit sparse_array = sparse.csr_matrix(array)
sparse_array.multiply(np.where(row_mask, 5, 1)[:, None]).tocsr()
```

```
340 µs ± 7.32 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

```
%%timeit sparse_array = sparse.csr_matrix(array)
sparse.diags(np.where(row_mask, 5, 1)).dot(sparse_array)
```

```
400 µs ± 6.43 µs per loop (mean ± std. dev. of 7 runs, 1000 loops each)
```

Fast for-loops using Numba

As a last resort, if there's no way to avoid a for-loop even with NumPy's vectorization capabilities, you can use the `@njit` decorator provided by the Numba package:

```
from numba import njit


@njit
def sum_array(arr):
    result = 0.0
    for i in range(arr.shape[0]):
        result += arr[i]
    return result
```

In fact, the Numba function is more than 100 times faster than without the decorator:

```
input_arr = np.float64(np.random.randint(low=0, high=10, size=(10000,)))
```

```
%timeit sum_array(input_arr)
```

```
10.9 µs ± 444 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

```
# Call the function without the @njit
%timeit sum_array.py_func(input_arr)
```

```
1.84 ms ± 65.4 µs per loop (mean ± std. dev. of 7 runs, 100 loops each)
```

However, whenever available, **NumPy's own vectorized functions will usually be faster than Numba**.

```
%timeit np.sum(input_arr)
%timeit input_arr.sum()
%timeit np.einsum("i->", input_arr)
```

```
7.6 µs ± 687 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
5.27 µs ± 411 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
7.89 µs ± 499 ns per loop (mean ± std. dev. of 7 runs, 100000 loops each)
```

Make sure you understand the basic idea behind Numba before using it, read the Numba docs.

Don't use `@jit`, but use `@njit` which is an alias for `@jit(nopython=True)`.

When you know what you are doing, the `fastmath` and `parallel` options can boost performance even further: read more about this in the Numba docs.

Parallelizing tasks

Depending on your hardware setup, parallelizing tasks using pathos and Numba's automatic parallelization feature can improve the performance of your implementation.

 Expensive hardware is no excuse for inefficient code.

Many tasks in CLIMADA could profit from GPU implementations. However, currently there are **no plans to include GPU support in CLIMADA** because of the considerable development and maintenance workload that would come with it. If you want to change this, contact the core team of developers, open an issue or mention it in the bi-weekly meetings.

Read NetCDF datasets with `xarray`

When dealing with NetCDF datasets, memory is often an issue, because even if the file is only a few megabytes in size, the uncompressed raw arrays contained within can be several gigabytes large (especially when data is sparse or similarly structured). One way of dealing with this situation is to open the dataset with `xarray`.

 `xarray` allows to read the shape and type of variables contained in the dataset without loading any of the actual data into memory.

Furthermore, when loading slices and arithmetically aggregating variables, memory is allocated not more than necessary, but values are obtained on-the-fly from the file.

⚠ Note that opening a dataset should be done with a context manager, to ensure proper closing of the file:

```
with xr.open_dataset("saved_on_disk.nc") as ds:
```

Using pickle to save python objects

`pickle` is the standard python library serialization module. It has the nice feature of being able to save most of python objects (standard and user defined) using simple methods. However, pickle is *transient*, i.e. it has limited Portability: Pickle files are specific to the Python environment they were created in. This means that Pickle files may not be compatible across different Python versions or environments, which can make it challenging to share data between systems. As such it should only be used for temporary storage and not for persistent one.

Take-home messages

We conclude by repeating the gist of this guide:

 Use profiling tools to find and assess performance bottlenecks.
 Replace for-loops by built-in functions and efficient external implementations.
 Consider algorithmic performance, not only implementation performance.
 Get familiar with NumPy: vectorized functions, slicing, masks and broadcasting.
- **Miscellaneous:** sparse arrays, Numba, parallelization, huge files (xarray), memory.
⚠ **Don't over-optimize** at the expense of readability and usability.

3.10 CLIMADA coding conventions

3.10.1 Dependencies (python packages)

Python is extremely powerful thanks to the large amount of available libraries, packages and modules. However, maintaining a code with a large number of such packages creates dependencies which is very care intensive. Indeed, each package developer can and does update and develop continuously. This means that certain code can become obsolete over time, stop working altogether, or become incompatible with other packages. Hence, it is crucial to keep the philosophy:

As many packages as needed, as few as possible.

Thus, when you are coding, follow these priorities:

1. Python standard library
2. Functions and methods already implemented in CLIMADA (do NOT introduce circulary imports though)
3. Packages already included in CLIMADA
4. Before adding a new dependency:
   - Contact a repository admin to get permission
   - Open an issue

Hence, first try to solve your problem with the standard library and function/methods already implemented in CLIMADA then use the packages included in CLIMADA, and if this is not enough, propose the addition of a new package. Do not hesitate to propose new packages if this is needed for your work!

3.10.2 Class inheritance

In Python, a class can inherit from other classes, which is a very useful mechanism in certain circumstances. However, it is wise to think about inheritance before implementing it. Very important to note, that CLIMADA classes DO NOT inherit from external library classes. For example, if `Exposure` class is directly inherited from the external package `Geopandas`, it may cause problems in CLIMADA if `Geopandas` is updated.

CLIMADA classes shall NOT inherit classes from external modules.

3.10.3 Avoid attribute-style accesses

CLIMADA developers shall use item-style access instead of attribute-style access (e.g. centroids.gdf["dist_coast"] instead of centroids.gdf.dist_coast) when accessing a column (in the example: "dist_coast") in a DataFrame or GeoDataFrame, or variables and attributes of xarray Datasets and DataArrays.

Reasons are: Improved syntax highlighting, more consistency (since in many cases you cannot use attribute-style access, so you are forced to fall back to item-style access), avoid mixing up attribute and column names.

3.10.4 Code formatting

Consistent code formatting is crucial for any project, especially open-source ones. It enhances readability, reduces cognitive load, and makes collaboration easier by ensuring that code looks the same regardless of who wrote it. Uniform formatting helps avoiding unnecessary differences in version control, focusing reviewson functional changes rather than stylistic differences.

Pull requests checks

Currently, the CI/CD pipeline checks that:

1. Every files end with a newline
2. There are no trailing whitespace at the end of lines.
3. All `.py` and `.ipynb` files are formatted following `black` convention
4. Import statements are sorted following `isort` convention

Note that most text editors usually take care of 1. and 2. by default.

Please note that pull requests will not be merged if these checks fail. The easiest way to ensure this, is to use *pre-commit hooks*, which will allow you to both run the checks and apply fixes when creating a new commit. Following the *advanced installation instructions* will set up these hooks for you.

black

We chose black as our formatter because it perfectly fits this need, quoting directly from the project

> Black is the uncompromising Python code formatter. By using it, you agree to cede control over minutiae of hand-formatting. In return, Black gives you speed, determinism, and freedom from pycodestyle nagging about formatting. You will save time and mental energy for more important matters. Blackened code looks the same regardless of the project you're reading. Formatting becomes transparent after a while and you can focus on the content instead. Black makes code review faster by producing the smallest diffs possible.

`black` automatically reformats your Python code to conform to the PEP 8 style guide, among other guidelines. It takes care of various aspects, including:

- Line Length: By default, it wraps lines to 88 characters, though this can be adjusted.
- Indentation: Ensures consistent use of 4 spaces for indentation.
- String Quotes: Converts all strings to use double quotes by default.
- Spacing: Adjusts spacing around operators and after commas to maintain readability.

For installation and more in-depth information on black, refer to its documentation.

Plugins executing `black` are available for our recommended IDEs:

- VSCode: Black Formatter Plugin
- Spyder: See this SO post
- JupyterLab: Code Formatter Plugin

isort

`isort` is a Python utility to sort imports alphabetically, and automatically separated into sections and by type.

Just like `black` it ensure consistency of the code, focusing on the imports

For installation and more in depth information on `isort` refer to its documentation.

A VSCode plugin is available.

How do I update my branch if it is not up to date with the formatted Climada?

If you were developing a feature before Climada switched to `black` formatting, you will need to follow a few steps to update your branch to the new formatting.

Given a feature branch `YOUR_BRANCH`, do the following:

1. Update the repo to fetch the latest changes:

```
git fetch -t
git checkout develop-white
git checkout develop-black
```

2. Switch to your feature branch and merge `develop-white` (in order to get the latest changes in `develop` before switching to `black`):

```
git checkout YOUR_BRANCH
git pull
pre-commit uninstall || pip install pre-commit
git merge --no-ff develop-white
```

If merge conflicts arise, resolve them and conclude the merge as instructed by Git. It also helps to check if the tests pass after the merge.

3. Install and run the pre-commit hooks:

```
pre-commit install
pre-commit run --all-files
```

4. Commit the changes applied by the hooks to your branch:

```
git add -u
git commit
```

5. Now merge `develop-black`:

```
git merge --no-ff develop-black
```

Resolve all conflicts by choosing "Ours" over "Theirs" ("Current Change" over the "Incoming Change").

```
git checkout --ours .
git add -u
git commit
```

6. Now, get up to date with the latest `develop` branch:

```
git checkout develop
git pull
git checkout YOUR_BRANCH
git merge --no-ff develop
```

Again, fix merge conflicts if they arise and check if the tests pass. Accept the incoming changes for the tutorials 1_main, Exposures, LitPop Impact, Forecast and TropicalCyclone unless you made changes to those. Again, the file with the most likely merging conflicts is CHANGELOG.md, which should probably be resolved by accepting both changes.

7. Finally, push your latest changes:

```
git push origin YOUR_BRANCH
```

3.10.5 Paper repository

Applications made with CLIMADA which are published in the form of a paper or a report are very much encouraged to be submitted to the climada/paper repository. You can either:

- Prepare a well-commented jupyter notebook with the code necessary to reproduce your results and upload it to the climada/paper repository. Note however that the repository cannot be used for storing data files.
- Upload the code necessary to reproduce your results to a separate repository of your own. Then, add a link to your repository and to your publication to the readme file on the climada/paper repository.

Notes about DOI

Some journals require you to provide a DOI to the code and data used for your publication. In this case, we encourage you to create a separate repository for your code and create a DOI using Zenodo or any specific service from your institution (e.g. ETH Zürich).

The CLIMADA releases are also identified with a DOI.

3.10.6 Utility functions

In CLIMADA, there is a set of utility functions defined in `climada.util`. A few examples are:

- convert large monetary numbers into thousands, millions or billions together with the correct unit name
- compute distances
- load hdf5 files
- convert iso country numbers between formats
- …

Whenever you develop a module or make a code review, be attentive to see whether a given functionality has already been implemented as a utility function. In addition, think carefully whether a given function/method does belong in its module or is actually independent of any particular module and should be defined as a utility function.

It is very important to not reinvent the wheel and to avoid unnecessary redundancies in the code. This makes maintenance and debugging very tedious.

3.10.7 Data dependencies

Web APIs

CLIMADA relies on open data available through web APIs such as those of the World Bank, Natural Earth, NASA and NOAA. You might execute the test `climada_python-x.y.z/test_data_api.py` to check that all the APIs used are active. If any is out of service (temporarily or permanently), the test will indicate which one.

Manual download

As indicated in the software and tutorials, other data might need to be downloaded manually by the user. The following table shows these last data sources, their version used, its current availability and where they are used within CLIMADA:

Name	Version	Link	CLIMADA class	CLIMADA version	CLIMADA tutorial reference
Fire Information for Resource Management System		FIRMS	BushFire	> v1.2.5	climada_hazard_BushFire.ipynb
Gridded Population of the World (GPW)	v4.11	GPW4.1	LitPop	> v1.2.3	climada_entity_LitPop.ipynb

3.10.8 Side note on parameters

Don't use *args and **kwargs parameters without a very good reason.

There *are* valid use cases for this kind of parameter notation.
In particular `*args` comes in handy when there is an unknown number of equal typed arguments to be passed. E.g., the `pathlib.Path` constructor.
But if the parameters are expected to be structured in any way, it is just a bad idea.

```
def f(x, y, z):
    return x + y + z


# bad in most cases
def g(*args, **kwargs):
    x = args[0]
```

(continues on next page)

(continued from previous page)

```
    y = kwargs["y"]
    s = f(*args, **kwargs)
    print(x, y, s)


g(1, y=2, z=3)
```

```
# usually just fine
def g(x, y, z):
    s = f(x, y, z)
    print(x, y, s)


g(1, y=2, z=3)
```

Decrease the number of parameters.

Though CLIMADA's pylint configuration .pylintrc allows 7 arguments for any method or function before it complains, it is advisable to aim for less. It is quite likely that a function with so many parameters has an inherent design flaw.

There are very well designed command line tools with inumerable optional arguments, e.g., rsync - but these are command line tools. There are also methods like `pandas.DataFrame.plot()` with countless optional arguments and it makes perfectly sense.

But within the climada package it probably doesn't. divide et impera!

Whenever a method has more than 5 parameters, it is more than likely that it can be refactored pretty easily into two or more methods with less parameters and less complexity:

```
def f(a, b, c, d, e, f, g, h):
    print(f"f does many things with a lot of arguments: {a, b, c, d, e, f, g, h}")
    return sum([a, b, c, d, e, f, g, h])


f(1, 2, 3, 4, 5, 6, 7, 8)
```

```
def f1(a, b, c, d):
    print(f"f1 does less things with fewer arguments: {a, b, c, d}")
    return sum([a, b, c, d])


def f2(e, f, g, h):
    print(f"f2 dito: {e, f, g, h}")
    return sum([e, f, g, h])


def f3(x, y):
    print(f"f3 dito, but on a higher level: {x, y}")
    return sum([x, y])


f3(f1(1, 2, 3, 4), f2(5, 6, 7, 8))
```

This of course pleads the case on a strictly formal level. No real complexities have been reduced during the making of this example.
Nevertheless there is the benefit of reduced test case requirements. And in real life, real complexity *will* be reduced.

3.11 Constants and Configuration

3.11.1 Constants

Constants are values that, once initialized, are never changed during the runtime of a program. In Python constants are assigned to variables with capital letters by convention, and vice versa, variables with capital letters are supposed to be constants.

In principle there are about four ways to define a constant's value:

- *hard coding*: the value is defined in the python code directly
- *argument*: the value is taken from an execution argument
- *context*: the value is derived from the environmental context of the execution, e.g., the current working directory or the date-time of execution start.
- *configuration*: read from a file or database

In CLIMADA, we only use *hard coding* and *configuration* to assign values to constants.

Hard Coded

Hard coding constants is the preferred way to deal with strings that are used to identify objects or files.

```
# suboptimal
my_dict = {"x": 4}
if my_dict["x"] > 3:
    msg = "well, arh, ..."
msg
```

```
'well, arh, ...'
```

```
# good
X = "x"
my_dict = {X: 4}
if my_dict[X] > 3:
    msg = "yeah!"
msg
```

```
'yeah!'
```

```
# possibly overdoing it
X = "x"
Y = "this doesn't mean that every string must be a constant"
my_dict = {X: 4}
if my_dict[X] > 3:
    msg = Y
msg
```

```
"this doesn't mean that every string must be a constant"
```

```
import pandas as pd

X = "x"
df = pd.DataFrame({"x": [1, 2, 3], "y": [4, 5, 6]})
try:
    df.X
except:
    from sys import stderr

    stderr.write("this does not work\n")
df[X]  # this does work but it's less pretty
df.x
```

```
this does not work
```

```
0    1
1    2
2    3
Name: x, dtype: int64
```

Configurable

When it comes to absolute paths, it is urgently suggested to not use hard coded constant values, for obvious reasons. But also relative paths can cause problems. In particular, they may point to a location where the user has not sufficient access permissions. In order to avoid these problems, *all* paths constants in CLIMADA are supposed to be defined through configuration.
→ paths must be configurable

The same applies to urls to external resources, databases or websites. Since they may change at any time, their addresses are supposed to be defined through configuration. Like this it will be possible to access them without the need of tampering with the source code or waiting for a new release.
→ urls must be configurable

Another category of constants that should go into the configuration file are system specifications, such as number of CPU's available for CLIMADA or memory settings.
→ OS settings must be configurable

Where to put constants?

As a general rule, constants are defined in the module where they intrinsically belong to. If they belong equally to different modules though or they are meant to be used globally, there is the module `climada.util.constants` which is compiling constants CLIMADA-wide.

3.11.2 Configuration

Configuration files

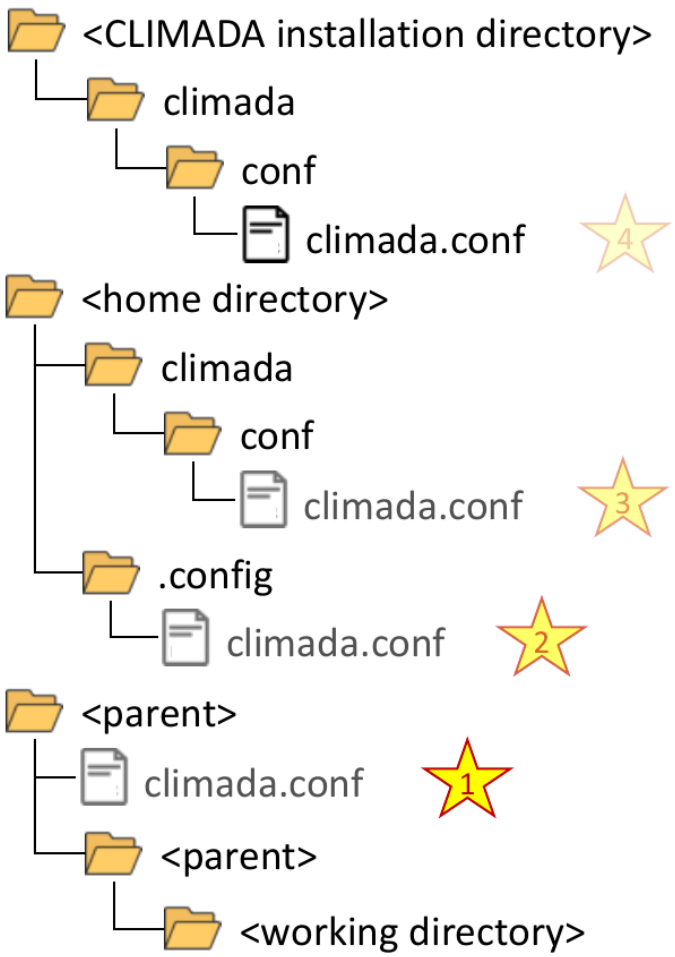
The proper place to define constants that a user may want (or need) to change without changing the CLIMADA installation are the configuration files.
These are files in *json* format with the name `climada.conf`. There is a default config file that comes with the installation of CLIMADA. But it's possible to have several of them. In this case they are complementing one another.

CLIMADA looks for configuration files upon `import climada`. There are four locations to look for configuration files:

- `climada/conf`, the installation directory
- `~/climada/conf`, the user's default climada directory
- `~/.config`, the user's configuration directory,
- `.`, the current working directory

At each location, the path is followed upwards until a file called `climada.conf` is found or the root of the path is reached. Hence, if e.g., `~/climada/climada.conf` is missing but `~/climada.conf` is present, the latter would be read.

When two config files are defining the same value, the priorities are:
`[..]/./climada.conf` > `~/.config/climada.conf` > `~/climada/conf/climada.conf` > `installation_dir/climada/conf/climada.conf`



Format

A configuration file is a JSON file, with the additional restriction, that all keys must be strings without a '.' (dot) character .

The JSON format looks a lot like a Python `dict`. But note, that all strings must be surrounded by double quotes and trailing commas are not allowed anywhere.

For configuration values that belong to a particular module it is suggested to reflect the code repositories file structure in

the json object. For example, if a configuration for `my_config_value` that belongs to the module `climada.util.dates_times` is wanted, it would be defined as

```
{
  "util": {
    "dates_times": {
      "my_config_value": 42
    }
  }
}
```

Referenced Configuration Values

Configuration string values can be referenced from other configuration values. E.g.

```
{
  "a": "x",
  "b": "{a}y"
}
```

In this example "b" is eventually resolved to "xy".

Accessing configuration values

Configuration values can be accessed through the (constant) `CONFIG` from the `climada` module:

```
from climada import CONFIG
```

```
CONFIG.hazard
```

```
{trop_cyclone: {random_seed: 54}, storm_europe: {forecast_dir: ./results/forecast/
↪hazards}, test_data: .../climada/hazard/test/data}
```

Data Types

The configuration itself and its attributes have the data type `climada.util.config.Config`

```
CONFIG.__class__, CONFIG.hazard.trop_cyclone.random_seed.__class__
```

```
(climada.util.config.Config, climada.util.config.Config)
```

The actual configuration values can be accessed as basic types (bool, float, int, str), provided that the definition is according to the respective data type:

```
CONFIG.hazard.trop_cyclone.random_seed.int()
```

```
54
```

```
try:
    CONFIG.hazard.trop_cyclone.random_seed.str()
except Exception as e:
    from sys import stderr
```

(continues on next page)

(continued from previous page)

```
    stderr.write(f"cannot convert random_seed to str: {e}\n")
```

```
cannot convert random_seed to str: <class 'int'>, not str
```

However, configuration string values can be converted to `pathlib.Path` objects if they are pointing to a directory.

```
CONFIG.hazard.storm_europe.forecast_dir.dir()
```

Note that converting a configuration string to a `Path` object like this will create the specified directory on the fly, unless `dir` is called with the parameter `create=False`.

Default Configuration

The conifguration file `climada/conf/climada.conf` contains the default configuration.
On the top level it has the following attributes:

- **local_data**: definition of main paths for accessing and storing CLIMADA related data
  - **system**: top directory, where (persistent) climada data is stored
    default: `~/climada/data`
  - **demo**: top directory for data that is downloaded or created in the CLIMADA tutorials
    default: `~/climada/demo/data`
  - **save_dir**: directory where transient (non-persistent) data is stored
    default: `./results`
- **log_level**: minimum log level showed by logging, one of DEBUG, INFO, WARNING, ERROR or CRITICAL.
  default: `INFO`
- **max_matrix_size**: maximum matrix size that can be used, can be decreased in order to avoid memory issues
  default: `100000000` (1e8)
- **exposures**: exposures modules specific configuration
- **hazard**: hazard modules specific configuration

```
CONFIG.__dict__.keys()
```

```
dict_keys(['_root', '_comment', 'local_data', 'engine', 'exposures', 'hazard', 'util',
↪ 'log_level', 'max_matrix_size', 'data_api', 'test_directory', 'test_data', 'disc_
↪rates', 'impact_funcs', 'measures'])
```

Test Configuration

The configuration values for unit and integration tests are not part of the *default configuration*, since they are irrelevant for the regular CLIMADA user and only aimed for developers.
The default test configuration is defined in the `climada.conf` file of the installation directory. This file contains paths to files that are read during tests. If they are part of the GitHub repository, their path i.g. starts with the `climada` folder within the installation directory:

```
{
    "_comment": "this is a climada configuration file meant to supersede the default␣
↪configuration in climada/conf during test",
    "test_directory": "./climada",
```

(continues on next page)

(continued from previous page)

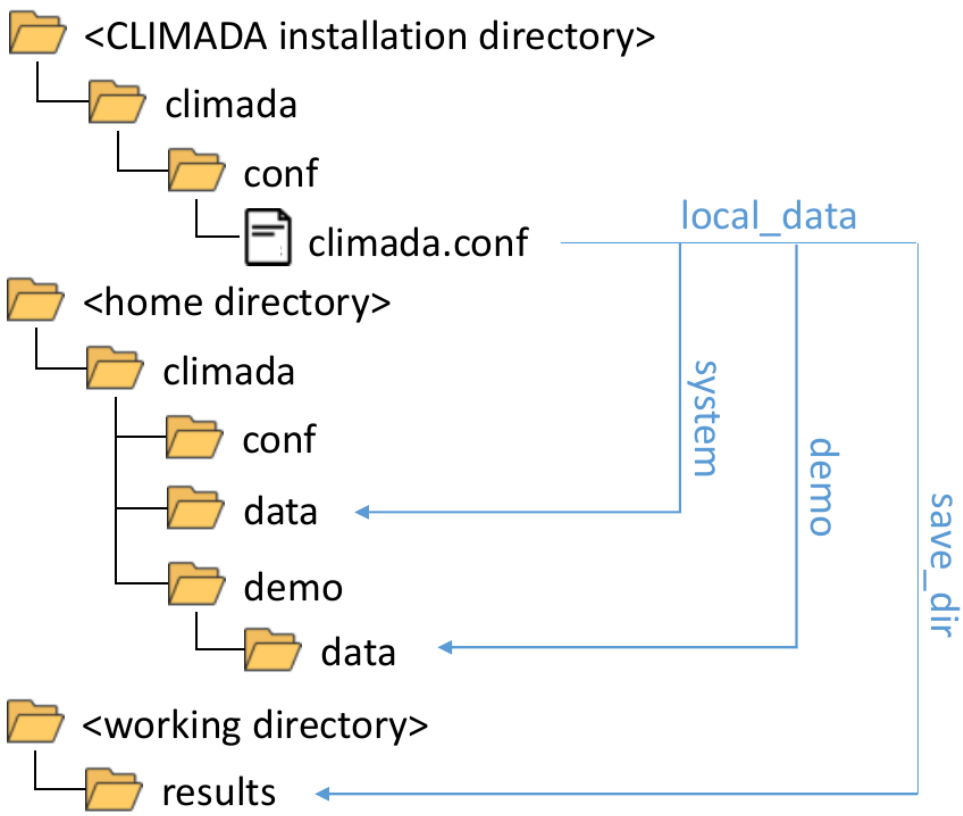
```
    "test_data": "{test_directory}/test/data",
    "disc_rates": {
        "test_data": "{test_directory}/entity/disc_rates/test/data"
    }
}
```

Obviously, the default `test_directory` is given as the relative path to `./climada`. This is fine if (but only if) unit or integration tests are started from the installation directory, which is the case in the automated tests on the CI server. Developers who intend to start a test from another working directory may have to edit this file and replace the relative path with the absolute path to the installation directory:

```
{
    "_comment": "this is a climada configuration file meant to supersede the default␣
↪configuration in climada/conf during test",
    "test_directory": "/path/to/installation-dir/climada",
    "test_data": "{test_directory}/test/data",
    "disc_rates": {
        "test_data": "{test_directory}/entity/disc_rates/test/data"
    }
}
```

Data Initialization

When `import climada` is executed in a python script or shell, data files from the installation directory are copied to the location specified in the current configuration.
This happens only when climada is used for the first time with the current configuration. Subsequent execution will only check for presence of files and won't overwrite existing files.



Thus, the home directory will automatically be populated with a climada directory and several files from the repository when climada is used.
To prevent this and keep the home directory clean, create a config file `~/.config/climada.conf` with customized values for `local_data.system` and `local_data.demo`.
As an example, a file with the following content would suppress creation of directories and copying of files during execution of CLIMADA code:

```
{
    "local_data": {
        "system": "/path/to/installation-dir/climada/data/system",
        "demo": "/path/to/installation-dir/climada/data/demo"
    }
}
```

3.12 Documentation writing

3.12.1 CLIMADA Tutorial Template

Why tutorials

Main goal:

The main goal of the tutorials is it to give a complete overview on:

- essential CLIMADA components

- introduce newly developed modules and features

More specifically, tutorials should introduce CLIMADA users to the core functionalities and modules and guide users in their application. Hence, each new module created needs to be accompanied with a tutorial. The following sections give an overview of the basic structure desired for CLIMADA tutorials.

Important:

A tutorial needs to be included with the final pull request for every new feature.

Basic structure

Every tutorial should cover the following main points. Additional features characteristic to the modules presented can and should be added as seen fit.

Introduction

- What is the feature presented?
  Briefly describe the feature and introduce how it's presented in the CLIMADA framework.
- What is its data structure?
  Present and overview (in the form of a table for example) of where the feature is built into CLIMADA. What class does it belong to, what are the variables of the feature, what is their data structure.
- Table of content:
  How is this tutorial structured?

Illustration of feature functionality and application

Walk users through the core functions of the module and illustrate how the feature can be used. This obviously is dependent on the feature itself. A few core points should be considered when creating the tutorial:

- **SIZE MATTERS!**
  - each notebook as a total should not exceed the critical (yet vague) size of "a couple MB"
  - keep the size of data you use as examples in the tutorial in mind
  - we aim for computational efficiency
  - a lean, well-organized, concise notebook is more informative than a long, messy all-encompassing one.
- follow the general CLIMADA naming convention for the notebook. For example: "climada_hazard_TropCyclone.ipynb"

Good examples

The following examples can be used as templates and inspiration for your tutorial:

- Exposure tutorial
- Hazard tutorial

Use only Markdown for headers and table of content

To create headers or a table of content with links, avoid using *hmtl* and prefer instead purely *Markdown* syntax. Follow *Markdown* conventions in the Markdown Guide and the following key points presented in the section below to know how to use correct *Markdown* syntax which is consistent with the rest of CLIMADA documentation. If in doubt, check existing tutorials to see how it is done.

Headers

To structure your tutorial, use headers of different levels to create sections and subsections.

To create an header, write the symbol (#) before your header name

'#' : create a header of level 1

'##' : create a header of level 2

'###' : create a header of level 3

'####' : create a header of level 4

The title of the tutorial should be of level 1 (#), should have its own cell, and should be the first cell of the notebook.

3.12.2 CLIMADA Documentation

The CLIMADA documentation consists of `.rst` files and Jupyter Notebooks. It is built into an HTML webpage by the Sphinx package.

The online documentation is automatically built when the `main` or `develop` branch are updated. Additionally, documentation previews will be built for every pull request on GitHub, and will be displayed under "Checks".

Note that the online documentation allows you to switch versions. By default, you will see the `stable` version, which refers to the latest release. You can switch to `latest`, which refers to the latest version of the `develop` branch.

Local Build

You can also build and browse the documentation on your machine. This can be useful if you want to access the documentation of a particular feature branch or to check your updates to the documentation.

For building the documentation, you need to follow the advanced installation instructions. Make sure to install the developer requirements as well.

Then, activate the `climada_env` and navigate to the `doc` directory:

```
conda activate climada_env
cd climada_python/doc
```

Next, execute `make` (this might take a while when executed for the first time)

```
make html
```

The documentation will be placed in `doc/_build/html`. Simply open the page `doc/_build/html/index.html` with your browser.

Updating the Documentation Environment for Readthedocs.org

The online documentation is built by `readthedocs.org`. Their servers have a limited capacity. In the past, this capacity was exceeded by Anaconda when it tried to resolve all dependencies for CLIMADA. We therefore provided a dedicated environment with *fixed* package versions in `requirements/env_docs.yml`. As of commit `8c66d8e4a4c93225e3a337d8ad69ab09b48278e3`, this environment was removed and the online documentation environment is built using the specs in `requirements/env_climada.yml`. If this should fail in the future, revert the changes by `8c66d8e4a4c93225e3a337d8ad69ab09b48278e3` and update the environment specs in `requirements/env_docs.yml` with the following instructions.

For re-creating the documentation environment, we provide a Dockerfile. You can use it to build a new environment and extract the exact versions from it. This might be necessary when we upgrade to a new version of Python, or when dependencies are updated. **NOTE:** Your machine must be able to run/virtualize an AMD64 OS.

Follow these instructions:

1. Install Docker on your machine.
2. Enter the top-level directory of the CLIMADA repository with your shell:

```
cd climada_python
```

3. Instruct Docker to build an image from the `doc/create_env_doc.dockerfile`:

```
docker build -f doc/create_env_doc.dockerfile -t climada_env_doc ./
```

4. Run a container from this image:

```
docker run -it climada_env_doc
```

5. You have now entered the container. Activate the conda environment and export its specs:

```
conda activate climada_doc
conda env export
```

   Copy and paste the shell output of the last command into the `requirements/env_docs.yml` file in the CLIMADA repository, overwriting all its contents.

3.13 Testing

3.13.1 Notes on Testing

Any programming code that is meant to be used more than once should have a test, i.e., an additional piece of programming code that is able to check whether the original code is doing what it's supposed to do.

Writing tests is work. As a matter of facts, it can be a *lot* of work, depending on the program often more than writing the original code.
Luckily, it essentially follows always the same basic procedure and a there are a lot of tools and frameworks available to facilitate this work.

In CLIMADA we use the Python in-built *test runner* pytest for execution of the tests.

Why do we write test?

- The code is most certainly **buggy** if it's not properly tested.
- Software without tests is **worthless**. It won't be trusted and therefore it won't be used.

When do we write test?

- **Before implementation.** A very good idea. It is called Test Driven Development.
- **During implementation.** Test routines can be used to run code even while it's not fully implemented. This is better than running it interactively, because the full context is set up by the test.
  By command line:
  `python -m unittest climada.x.test_y.TestY.test_z`
  Interactively:
  `climada.x.test_y.TestY().test_z()`
- **Right after implementation.** In case the coverage analysis shows that there are missing tests, see *Test Coverage*.
- **Later, when a bug was encountered.** Whenever a bug gets fixed, also the tests need to be adapted or amended.

Basic Test Procedure

- **Test data setup**
  Creating suitable test data is crucial, but not always trivial. It should be extensive enough to cover all functional requirements and yet as small as possible in order to save resources, both in space and time.
- **Code execution**
  The main goal of a test is to find bugs *before* the user encounters them. Ultimately every single line of the program should be subject to test.
  In order to achieve this, it is necessary to run the code with respect to the whole parameter space. In practice that means that even a simple method may require a lot of test code.
  (Bear this in mind when designing methods or functions: the number of required tests increases dramatically with the number of function parameters!)
- **Result validation**
  After the code was executed the *actual* result is compared to the *expected* result. The expected result depends on test data, state and parametrization.
  Therefore result validation can be very extensive. In most cases it won't be practical nor required to validate every single byte. Nevertheless attention should be paid to validate a range of results that is wide enough to discover as many thinkable discrepancies as possible.

Testing types

Despite the common basic procedure there are many different kinds of tests distinguished. (See WikiPedia:Software testing). Very commonly a distinction is made based on levels:

- **Unit Test**: tests only a small part of the code, a single function or method, essentially without interaction between modules
- **Integration Test**: tests whether different methods and modules work well with each other
- **System Test**: tests the whole software at once, using the exposed interface to execute a program

Unit Tests

Unit tests are meant to check the correctness of program units, i.e., single methods or functions, they are supposed to be fast, simple and easy to write.

Developer guidelines:

- **Each module in CLIMADA has a counter part containing unit tests.**
  Naming suggestion: `climada.x.y` → `climada.x.test.test_y`
- **Write a test class for each class of the module, plus a test class for the module itself in case it contains (module) functions.**
  Naming suggestion: `class X` → `class TestX(unittest.TestCase)`, module `climda.x.y` → `class TestY(unittest.TestCase)`
- **Ideally, each method or function should have at least one test method.**
  Naming suggestion: `def xy()` → `def test_xy()`, `def test_xy_suffix1()`, `def test_xy_suffix2()`
  Functions that are created for the sole purpose of structuring the code do not necessarily have their own unit test.
- **Aim at having *very* fast unit tests!**
  There will be hundreds of unit tests and in general they are called in corpore *and expected to finish after a reaonable amount of time.*
  Less than 10 milisecond is good, 2 seconds is the maximum acceptable duration.
- **A unit test shouldn't call more than one climada method or function.**
  The motivation to combine more than one method in a test is usually creation of test data. Try to provide test data by

other means. Define them on the spot (within the code of the test module) or create a file in a test data directory that can be read during the test. If this is too tedious, at least move the data acquisition part to the constructor of the test class.

- **Do not use external resources in unit tests.**
  Methods depending on external resources can be skipped from unit tests.

Integration Tests

Integration tests are meant to check the correctness of interaction between units of a module or a package.
As a general rule, more work is required to write integration tests than to write unit tests and they have longer runtime.

Developer guidelines:

- **Write integration tests for all intended use cases**.
- **Do not expect external resources to be immutable**.
  If calling on external resources is part of the workflow to be tested, take into account that they may change over time.
  If the according API has means to indicate the precise version of the requested data, make use of it, otherwise, adapt your expectations and leave room for future changes.
  Example given: your function is ultimately relying on the *current* GDP retrieved from an online data provider, and you test it for Switzerland where it's in about 700 Bio CHF at the moment. Leave room for future development, try to be on a reasonably save side, tolerate a range between 70 Bio CHF and 7000 Bio CHF.
- **Test location**.
  Integration are written in modules `climada.test.test_xy` or in `climada.x.test.test_y`, like the unit tests.
  For the latter it is required that they do not use external resources and that the tests do not have a runtime longer than 2 seconds.

System Tests

System tests are meant to check whether the whole software package is working correctly.

In CLIMADA, the system test that checks the core functionality of the package is executed by calling `make install_test` from the installation directory.

Error Messages

When a test fails, make sure the raised exception contains all information that might be helpful to identify the exact problem.
If the error message is ever going to be read by someone else than you while still developing the test, you best assume it will be someone who is completely naive about CLIMADA.

Writing extensive failure messages will eventually save more time than it takes to write them.

Putting the failure information into logs is neither required nor sufficient: the automated tests are built around error messages, not logs.
Anything written to `stdout` by a test method is useful mainly for the developer of the test.

Test Coverage

Coverage is a measure of how much of your code is actually checked by the tests. One distinguishes between line coverage and branch or conditionals coverage. The line coverage reports the percentage of all lines of code covered by the tests. The branch coverage reports the percentage of all possible branches covered by the tests. Achieving a high branch coverage is much harder than a high line coverage.

In CLIMADA, we aim for a high line coverage (only). Ideally, any new code should have a line coverage of 100%, meaning every line of code is tested. You can inspect the test coverage of your local code by following the instructions for executing tests below.

See the *Continuous Integration Guide* for information on how to inspect coverage of the automated test pipeline.

Test files

For integration tests it can be required to read data from a file, in order to set up a test that aims to check functionality with non-trivial data, beyond the scope of unit tests. Some of thes test files can be found in the `climada/**/test/data` directories or in the `climada/data` directory. As mostly the case with large test data, it is not very well suitable for a Git repository.

The preferable alternative is to post the data to the Climada Data-API with status `test_dataset` and retrieve the files on the fly from there during tests. To do this one can use the convenience method `climada.test.get_test_file`:

```
from climada.test import get_test_file

my_test_file = get_test_file(
    ds_name="my-test-file", file_format="hdf5"
)  # returns a pathlib.Path object
```

Behind the scenes, `get_test_file` uses the `climada.util.api_client.Client` to identify the appropriate dataset and downloads the respective file to the local dataset cache (`~/climada/data/*`).

Dealing with External Resources

Methods depending on external resources (calls a url or database) are ideally atomic and doing nothing else than providing data. If this is the case they can be skipped in unit tests on safe grounds - provided they are tested at some point in higher level tests.

In CLIMADA there are the utility functions `climada.util.files_handler.download_file` and `climada.util.files_handler.download_ftp`, which are assigned to exactly this task for the case of external data being available as files.

Any other method that is calling such a data providing method can be made compliant to unit test rules by having an option to replace them by another method. Like this one can write a dummy method in the test module that provides data, e.g., from a file or hard coded, which be given as the optional argument.

```
import climada


def x(download_file=climada.util.files_handler.download_file):
    filepath = download_file("http://real_data.ch")
    return Path(filepath).stat().st_size


import unittest


class TestX(unittest.TestCase):
    def download_file_dummy(url):
        return "phony_data.ch"

    def test_x(self):
        self.assertEqual(44, x(download_file=self.download_file_dummy))
```

Developer guideline:

- When introducing a new external resource, add a test method in `test_data_api.py`.

Test Configuration

Use the configuration file `climada.config` in the installation directory to define file paths and external resources used during tests (see the *Constants and Configuration Guide*).

3.13.2 Testing CLIMADA

Executing the entire test suite requires you to install the additional requirements for testing. See the *installation instructions for developer dependencies* for further information.

In general, you execute tests with

```
pytest <path>
```

where you replace `<path>` with a Python file containing tests or an entire directory containing multiple test files. Pytest will walk through all subdirectories of `<path>` and try to discover all tests. For example, to execute *all tests* within the CLIMADA repository, execute

```
pytest climada/
```

from within the `climada_python` directory.

Installation Test

From the installation directory run

```
make install_test
```

It lasts about 45 seconds. If it succeeds, CLIMADA is properly installed and ready to use.

Unit Tests

From the installation directory run

```
make unit_test
```

It lasts about 5 minutes and runs unit tests for all modules.

Integration Tests

From the installation directory run

```
make integ_test
```

It lasts about 15 minutes and runs extensive integration tests, during which also data from external resources is read. An open internet connection is required for a successful test run.

Coverage

Executing `make unit_test` and `make integ_tests` provides local coverage reports as HTML pages at `coverage/index.html`. You can open this file with your browser.

3.14 Reviewer Guidelines

3.14.1 How to review a pull request

- Be friendly
- Read and follow the ***Reviewer Checklist***
- Decide how much time you can spare and the detail you can work in. Tell the author!
- Use the comment/chat functionality within GitHub's pull requests - it's useful to have an archive of discussions and the decisions made.
- Fix the big things first! If there are more important issues, not every style guide has to be stuck to, not every slight increase in speed needs to be pointed out, and test coverage doesn't have to be 100%.
- Make it clear when a change is optional, or is a matter of opinion

At a minimum

- Make sure unit and integration tests are passing.
- (For complete modules) Run the tutorial on your local machine and check it does what it says it does
- Check everything is fully documented

At least one reviewer needs to

- Review all the changes in the pull request. Read what it's supposed to do, check it does that, and make sure the logic is sound.
- Check that the code follows the CLIMADA style guidelines
- *CLIMADA coding conventions*
- *Python Dos and Don't*
- *Python performance tips and best practice for CLIMADA developers*
- If the code is implementing an algorithm it should be referenced in the documentation. Check it's implemented correctly.
- Try to think of edge cases and ways the code could break. See if there's appropriate error handling in cases where the function might behave unexpectedly.
- (Optional) suggest easy ways to speed up the code, and more elegant ways to achieve the same goal.

There are a few ways to suggest changes

- As questions and comments on the pull request page
- As code suggestions (max a few lines) in the code review tools on GitHub. The author can then approve and commit the changes from GitHub pull request page. This is great for typos and little stylistic changes.
- If you decide to help the author with changes, you can either push them to the same branch, or create a new branch and make a pull request with the changes back into the branch you're reviewing. This lets the author review it and merge.

3.14.2 Setup the environment

In order to make your review, you will have to have climada installed from source (see *here*), and switch to the branch you are reviewing (see *here*)

Creating a new python environment is often not necessary (*e.g.*, for minor feature branch that do not change the dependencies you can probably use a generic `develop` environment where you installed climada in editable mode), but can help in some cases (for instance changes in dependencies), to do so:

Here is a generic set of instructions which should always work, assuming you already cloned the climada repository, and are at the root of that folder:

```
git fetch && git checkout <branch to review>
mamba create -n <choose a name> # restrict python version here with "python==3.x.*"
mamba env update -n <same name> -f requirements/env_climada.yml
mamba activate <same name>
pip install -e ./
```

3.14.3 Reviewer Checklist

- The code must be readable without extra effort from your part. The code should be easily readable (for infos e.g. here)
- Include references to the used algorithms in the docstring
- If the algorithm is new, please include a description in the docstring, or be sure to include a reference as soon as you publish the work
- Variable names should be chosen to be clear. Avoid `item, element, var, list, data` etc… A good variable name makes it immediately clear what it contains.
- Avoid as much as possible hard-coded indices for list (no `x = l[0], y = l[1]`). Rather, use tuple unpacking (see here). Note that tuple unpacking can also be used to update variables. For example, the Fibonacci sequence next number pair can be written as `n1, n2 = n2, n1+n2`.
- Do not use mutable (lists, dictionaries, …) as default values for functions and methods. Do not write:

  ```
  def function(default=[]):
  ```

  but use

  ```
  def function(default=None):
      if default is None: default=[]
  ```

- Use pythonic loops, list comprehensions
- Make sure the unit tests are testing all the lines of the code. Do not only check for working cases, but also the most common wrong use cases.
- Check the docstrings (Do they follow the Numpydoc conventions, is everything clearly explained, are the default values given and is it clear why they are set to this value)
- Keep the code simple. Avoid using complex Python functionalities whose use is opaque to non-expert developers unless necessary. For example, the `@staticmethod` decorator should only be used if really necessary. Another example, for counting the dictionary `colors = ['red', 'green', 'red', 'blue', 'green', 'red']`, version:

  ```
  d = {}
  for color in colors:
          d[color] = d.get(color, 0) + 1
  ```

  is perfectly fine, no need to complicate it to a maybe more pythonic version

```
d = collections.defaultdict(int)
for color in colors:
    d[color] += 1
```

- Did the code writer perform a static code analysis? Does the code respect Pep8 (see also the pylint config file)?
- Did the code writer perform a profiling and checked that there are no obviously inefficient (computation time-wise and memory-wise) parts in the code?

3.15 Using Climada on the Euler Cluster (ETH internal)

3.15.1 Access to Euler

See https://scicomp.ethz.ch/wiki/Getting_started_with_clusters for details on how to register at and get started with Euler.

For all steps below, first enter the Cluster via SSH.

3.15.2 Installation directory and working directory

Please, get familiar with the various Euler storage options: https://scicomp.ethz.ch/wiki/Storage_systems. As a general rule: use `/cluster/project` for installation and `/cluster/work` for data processing.

For ETH WCR group members, the suggested installation and working directories are `/cluster/project/climate/$USER` and `/cluster/work/climate/$USER` respectively. You may have to create the installation directory:

```
mkdir -p /cluster/project/climate/$USER \
         /cluster/work/climate/$USER
```

3.15.3 Climada installation in a virtual environment

1. Load dependencies

```
module load \
  gcc/12.2.0 \
  stack/2024-06 \
  python/3.11.6 \
  hdf5/1.14.3 \
  geos/3.9.1 \
  sqlite/3.43.2 \
  eccodes/2.25.0 \
  gdal/3.6.3 \
  eth_proxy

module load proj
module unload proj
```

(The last two lines may seem odd but they are working around a conficting dependency version situation.)

You need to execute this every time you login to Euler before Climada can be used. To safe yourself from doing it manually, append these lines to the ~/.bashrc script, which is automatically executed upon logging in to Euler.

2. Create and prepare virtual environment

```
envname=climada_env

# create environment
python -m venv --system-site-packages /cluster/project/climate/$USER/venv/$envname

# acitvate it
. /cluster/project/climate/$USER/venv/$envname/bin/activate

# install python kernel (to be used in JupyterHub, s.b.)
python -m ipykernel install --user --name $envname
```

3. Install dependencies

```
pip install \
  dask[dataframe] \
  fiona==1.9 \
  gdal==3.6 \
  netcdf4==1.6.2 \
  rasterio==1.4 \
  pyproj==3.7 \
  geopandas==1.0 \
  xarray==2024.9 \
  sparse==0.15
```

4. Install Climada

There are two options. Either install from the downloaded repository (option A), or use a particular released version (option B).

option A

```
cd /cluster/project/climate/$USER  # or wherever you plan to download the repository
git clone https://github.com/CLIMADA-project/climada_python.git  # unless this has␣
↪been done before
cd climada_python
pip install -e .
```

If you need to work with a specific branch of Climada, you can do so by checking out to the target branch `your_branch` by running `git checkout your_branch` after having cloned the Climada repository and before running `pip install -e ..`

option B

```
pip install climada=5.0
```

or whatever version you prefer

5. Adjust the Climada configuration

Edit a configuration file according to your needs (see *Configuration*). Create a climada.conf file e.g., in /cluster/home/$USER/.config with the following content:

```
{
    "local_data": {
        "system": "/cluster/work/climate/USERNAME/climada/data",
        "demo": "/cluster/project/climate/USERNAME/climada/data/demo",
        "save_dir": "/cluster/work/climate/USERNAME/climada/results"
    }
}
```

(Replace USERNAME with your nethz-id.)

6. Test the installation

Check installation in login node:

```
python -m unittest climada.engine.test.test_impact_calc
```

This should prompt the usual "`OK`" in the end. Once that succeeded you may want to test the installation also in a compute node, just for the sake of it:

```
sbatch --wrap="python -m unittest climada.engine.test.test_impact_calc"
```

Look for the "`OK`" in the hereby created `slurm-[XXXXXXX].out` file

Please see the docs at https://slurm.schedmd.com/ on how to use the `slurm` batch system and the Wiki https://scicomp.ethz.ch/wiki/Transition_from_LSF_to_Slurm for a mapping of `lsf` commands to their `slurm` equivalents.

7. Optional: Install Climada Petals

To install Climada Petals, repeat the steps described in step 4A, but replacing the climada_python with the climada_petals repository:

```
cd /cluster/project/climate/$USER  # or wherever you plan to download the repository
git clone https://github.com/CLIMADA-project/climada_petals.git  # unless this has␣
↪been done before
cd climada_petals
pip install -e .
```

3.15.4 Run a Jupyter Notebook on Euler

It is possible to run a Jupyter Notebook on Euler within a JupytherHub instance running as an interactive slurm job. See the documentation https://scicomp.ethz.ch/wiki/JupyterHub.

For using climada inside the jupyter notebook, you need to create a customized `jupyterlabrc` file by running the following lines:

```
mkdir -p ~/.config/euler/jupyterhub
cat > ~/.config/euler/jupyterhub/jupyterlabrc <<EOF

module purge
```

(continues on next page)

(continued from previous page)

```
module load \
  gcc/12.2.0 \
  stack/2024-06 \
  python/3.11.6 \
  hdf5/1.14.3 \
  geos/3.9.1 \
  sqlite/3.43.2 \
  eccodes/2.25.0 \
  gdal/3.6.3 \
  r/4.3.2 \
  eth_proxy

module load proj

module unload proj

EOF
```

Check for the presence of a `~/.config/euler/jupyterhub/jupyterlab.sh` file or link. If it is missing, run

```
ln -s /cluster/software/others/services/jupyterhub/scripts/jupyterlab.sh ~/.config/
↪euler/jupyterhub/jupyterlab.sh
```

Now you can start a jupyter notebook running on Euler at https://jupyter.euler.hpc.ethz.ch/

Once the Jupyterlab has started, open a Python notebook and select the kernel `$envname`, which was created in step 2.

3.15.5 Trouble shooting

1. Python Module not found or available

- Make sure your python environment is activated.
- Run `pip install --upgrade MISSING_MODULE`.

2. Upgrading from Python 3.9 or 3.10

Virtual environments created are i.g. only working for the Python version they were created with. In particular Python kernels from 3.9 environments will fail to connect in a Jupyter notebook on https://jupyter.euler.hpc.ethz.ch/.

- It's suggested to create new environments and remove the old kernels from `~/.local/share/jupyter/kernels/`.

3. Incompatible GEOS version

If you get a warning `UserWarning: The Shapely GEOS version (3.9.1-CAPI-1.14.2) is incompatible with the GEOS version PyGEOS was compiled with (3.9.1-CAPI-1.14.2). Conversions between both will be slow.` or similar (version numbers may vary), updating geopandas can help:

- Create and activate a virtual environment with `venv` (s.a.)
- Run `pip install --upgrade geopandas`

4. Installation doesn't work

If you have additional requirements it may be that the installation process described above is failing. In this case you can run climada from a customized singularity container.

3.15.6 Fall back: Singularity Container

In case the installation in a virtual environment does not work, e.g,, because some module on Euler in incompatible with additional requirements for Python packages, the last resort is an installation of CLIMADA into a Singularity container. In general, this is more difficult and time-consuming and easier to get wrong. It also requires a lot of diskspace and produces a high number of files, but it provides more flexibility, as one can install basically anything you want.

To install CLIMADA into a Singularity container, follow these steps:

1. create a recipe

Create a file `recipe.txt` with the following content:

```
Bootstrap: docker
From: nvidia/cuda:12.0.0-devel-ubuntu22.04


%labels
    version="1.0.0"
    description="climada"


%post

    # Install requirements
    apt-get -y update
    DEBIAN_FRONTEND="noninteractive" TZ="Europe/Rome" apt-get -y install tzdata
    apt-get install -y `apt-cache depends openssh-client | awk '/Depends:/{print$2}'`
    apt-get download openssh-client
    dpkg --unpack openssh-client*.deb
    rm /var/lib/dpkg/info/openssh-client.postinst -f
    dpkg --configure openssh-client
    apt-get -y install tk tcl rsync wget curl git patch

    mkdir -p /opt/software

    # Install conda and mamba
    cd /opt/software
    curl -O https://repo.anaconda.com/miniconda/Miniconda3-latest-Linux-x86_64.sh
    sh ./Miniconda3-latest-Linux-x86_64.sh -p /opt/software/conda -b
    /opt/software/conda/bin/conda install -y -c conda-forge mamba

    # Create and activate environment
    /opt/software/conda/bin/mamba create -n climada_env python=3.11 --yes
    . /opt/software/conda/etc/profile.d/conda.sh && conda activate climada_env

    # Install jupyter
    python -m pip install jupyterhub jupyterlab
```

(continues on next page)

(continued from previous page)

```
    # Install climada from source
    mkdir -p /opt/climada_workspace
    cd /opt/climada_workspace

    git clone https://github.com/CLIMADA-project/climada_python.git
    cd climada_python
    git checkout develop

    mamba env update -n climada_env -f requirements/env_climada.yml
    python -m pip install -e ./

    # Install climada-petals from source
    cd /opt/climada_workspace

    git clone https://github.com/CLIMADA-project/climada_petals.git
    cd climada_petals
    git checkout develop

    mamba env update -n climada_env -f requirements/env_climada.yml
    python -m pip install -e ./


%environment
    #export LC_ALL=C


%runscript
   . /opt/software/conda/bin/activate && conda activate climada_env
   $@
```

2. build the container

- as it is cpu and memory consuming, run a job
- the container will be created in the directory `climada.sif`, it is going to be huge, so it's best located within the `project` file system

```
sbatch \
  --ntasks=1\
  --cpus-per-task=1 \
  --time=1:00:00 \
  --job-name="build-climada-container" \
  --mem-per-cpu=4096 \
  --wrap="singularity build --sandbox /cluster/project/[path/to]/climada.sif recipe.
↪txt"
```

3. Configure jupyterhub

create a file `~/.config/euler/jupyterhub/jupyterlabrc` with the following content:

```
#!/bin/bash
```

(continues on next page)

(continued from previous page)

```
# Import required modules
module purge
module load stack/2024-05 gcc/13.2.0 python/3.11.6_cuda eth_proxy


# Setup the required env
export JUPYTER_CONFIG_PATH=$HOME/.jupyterlab:$PYTHON_EULER_ROOT/share/jupyter
export JUPYTER_CONFIG_DIR=$HOME/.jupyterlab
export JUPYTER_PATH=$PYTHON_EULER_ROOT/share/jupyter
export JUPYTERLAB_DIR=$PYTHON_EULER_ROOT/share/jupyter/lab
export JUPYTERLAB_ROOT=$PYTHON_EULER_ROOT

export JUPYTER_HOME=${JUPYTER_HOME:-$HOME}
export JUPYTER_DIR=${JUPYTER_DIR:-/}
export JUPYTER_EXTRA_ARGS=${JUPYTER_EXTRA_ARGS:-}

warn_jupyterhub

sleep 1

echo $PYTHON_EULER_ROOT
echo $JUPYTER_EXTRA_ARGS
echo $PROXY_PORT

export PYTHON_ROOT=/opt/software/conda/envs/climada_env
module purge

export APPTAINER_BIND="/cluster,$TMPDIR,$SCRATCH"
singularity exec --nv \
  --env="NVIDIA_VISIBLE_DEVICES=all" \
  --bind /cluster/project/[path/to]/climada_python:/opt/climada_workspace/climada_
↪python \
  --bind /cluster/project/[path/to]/climada_petals:/opt/climada_workspace/climada_
↪petals \
  /cluster/project/[path/to]/climada.sif \
  /bin/bash <<EOF
. /opt/software/conda/bin/activate && conda activate climada_env
export JUPYTER_CONFIG_PATH=$HOME/.jupyterlab:$PYTHON_ROOT/share/jupyter
export JUPYTER_CONFIG_DIR=$HOME/.jupyterlab
export JUPYTER_PATH=$PYTHON_ROOT/share/jupyter
export JUPYTERLAB_DIR=$PYTHON_ROOT/share/jupyter/lab
export JUPYTERLAB_ROOT=$PYTHON_ROOT
export http_proxy=http://proxy.ethz.ch:3128
export https_proxy=http://proxy.ethz.ch:3128
jupyter lab build
jupyterhub-singleuser \
  --preferred-dir="$JUPYTER_HOME" \
  --notebook-dir="$JUPYTER_DIR" $JUPYTER_EXTRA_ARGS \
  --keyfile="$CONFIG_PATH/jupyter.key" \
  --certfile="$CONFIG_PATH/jupyter.crt" \
  --port="$PROXY_PORT"
EOF
```

replace the `[path/to]` bits according to your own directories.

Now you can start a jupyter notebook running on Euler at https://jupyter.euler.hpc.ethz.ch/

Once the Jupyterlab has started, open a Python notebook and stick to the default kernel.

3.16 CLIMADA List of Authors

- Gabriela Aznar-Siguan
- David N. Bresch
- Samuel Eberenz
- Jan Hartman
- Marine Perus
- Thomas Röösli
- Dario Stocker
- Veronica Bozzini
- Tobias Geiger
- Carmen B. Steinmann
- Evelyn Mühlhofer
- Rachel Bungerer
- Inga Sauer
- Samuel Lüthi
- Pui Man Kam
- Simona Meiler
- Alessio Ciullo
- Thomas Vogt
- Benoit P. Guillod
- Chahan Kropf
- Emanuel Schmid
- Chris Fairless
- Jan Wüthrich
- Zélie Stalhandske
- Yue Yu
- Lukas Riedel
- Raphael Portmann
- Nicolas Colombi
- Leonie Villiger
- Timo Schmid
- Kam Lam Yeung

- Sarah Hülsen
- Timo Schmid
- Luca Severino
- Samuel Juhel
- Valentin Gebhart
- Dahyann Araya

CHAPTER

FOUR

API REFERENCE

The API reference contains the whole specification of the code, that is, every modules, classes (and their attributes), and functions that are available (and documented).

4.1 Software documentation per package

4.1.1 climada.engine package

climada.engine.unsequa package

climada.engine.unsequa.calc_base module

class climada.engine.unsequa.calc_base.**Calc**

Bases: object

Base class for uncertainty quantification

Contains the generic sampling and sensitivity methods. For computing the uncertainty distribution for specific CLIMADA outputs see the subclass CalcImpact and CalcCostBenefit.

_input_var_names

Names of the required uncertainty variables.

Type
tuple(str)

_metric_names

Names of the output metrics.

Type
tuple(str)

Notes

Parallelization logics: for computation of the uncertainty users may specify a number N of processes on which to perform the computations in parallel. Since the computation for each individual sample of the input parameters is independent of one another, we implemented a simple distribution on the processes.

1. The samples are divided in N equal sub-sample chunks
2. Each chunk of samples is sent as one to a node for processing

Hence, this is equivalent to the user running the computation N times, once for each sub-sample. Note that for each process, all the input variables must be copied once, and hence each parallel process requires roughly the same amount of memory as if a single process would be used.

This approach differs from the usual parallelization strategy (where individual samples are distributed), because each sample requires the entire input data. With this method, copying data between processes is reduced to a minimum.

Parallelization is currently not available for the sensitivity computation, as this requires all samples simoultenaously in the current implementation of the SaLib library.

__init__()

Empty constructor to be overwritten by subclasses

check_distr()

Log warning if input parameters repeated among input variables

Return type
True.

property input_vars

Uncertainty variables

Returns
All uncertainty variables associated with the calculation

Return type
tuple(UncVar)

property distr_dict

Dictionary of the input variable distribution

Probabilitiy density distribution of all the parameters of all the uncertainty variables listed in self.InputVars

Returns
distr_dict – Dictionary of all probability density distributions.

Return type
dict(sp.stats objects)

est_comp_time(*n_samples*, *time_one_run*, *processes=None*)

Estimate the computation time

Parameters

- **n_samples** (*int/float*) – The total number of samples
- **time_one_run** (*int/float*) – Estimated computation time for one parameter set in seconds
- **pool** (*pathos.pool, optional*) – pool that would be used for parallel computation. The default is None.

Return type
Estimated computation time in secs.

make_sample(*N*, *sampling_method='saltelli'*, *sampling_kwargs=None*)

Make samples of the input variables

For all input parameters, sample from their respective distributions using the chosen sampling_method from SALib. https://salib.readthedocs.io/en/latest/api.html

This sets the attributes unc_output.samples_df, unc_output.sampling_method, unc_output.sampling_kwargs.

Parameters

- **N** (*int*) – Number of samples as used in the sampling method from SALib

- **sampling_method** (*str, optional*) – The sampling method as defined in SALib. Possible choices: 'saltelli', 'latin', 'morris', 'dgsm', 'fast_sampler', 'ff', 'finite_diff', https://salib.readthedocs.io/en/latest/api.html The default is 'saltelli'.
- **sampling_kwargs** (*kwargs, optional*) – Optional keyword arguments passed on to the SALib sampling_method. The default is None.

Returns
unc_output – Uncertainty data object with the samples

Return type
climada.engine.uncertainty.unc_output.UncOutput()

Notes

The 'ff' sampling method does not require a value for the N parameter. The inputed N value is hence ignored in the sampling process in the case of this method. The 'ff' sampling method requires a number of uncertainty parameters to be a power of 2. The users can generate dummy variables to achieve this requirement. Please refer to https://salib.readthedocs.io/en/latest/api.html for more details.

See also

SALib.sample
sampling methods from SALib SALib.sample

https
//salib.readthedocs.io/en/latest/api.html

sensitivity(*unc_output*, *sensitivity_method='sobol'*, *sensitivity_kwargs=None*)

Compute the sensitivity indices using SALib.

Prior to doing the sensitivity analysis, one must compute the uncertainty (distribution) of the output values (with self.uncertainty()) for all the samples (rows of self.samples_df).

According to Wikipedia, sensitivity analysis is "the study of how the uncertainty in the output of a mathematical model or system (numerical or otherwise) can be apportioned to different sources of uncertainty in its inputs." The sensitivity of each input is often represented by a numeric value, called the sensitivity index. Sensitivity indices come in several forms.

This sets the attributes: sens_output.sensistivity_method sens_output.sensitivity_kwargs sens_output.xxx_sens_df for each metric unc_output.xxx_unc_df

Parameters

- **unc_output** (*climada.engine.unsequa.UncOutput*) – Uncertainty data object in which to store the sensitivity indices
- **sensitivity_method** (*str, optional*) – Sensitivity analysis method from SALib.analyse. Possible choices: 'sobol', 'fast', 'rbd_fast', 'morris', 'dgsm', 'ff', 'pawn', 'rhdm', 'rsa', 'discrepancy', 'hdmr'. Note that in Salib, sampling methods and sensitivity analysis methods should be used in specific pairs: https://salib.readthedocs.io/en/latest/api.html
- **sensitivity_kwargs** (*dict, optional*) – Keyword arguments of the chosen SALib analyse method. The default is to use SALib's default arguments.

Notes

The variables 'Em','Term','X','Y' are removed from the output of the 'hdmr' method to ensure compatibility with unsequa. The 'Delta' method is currently not supported.

Returns
sens_output – Uncertainty data object with all the sensitivity indices, and all the uncertainty data copied over from unc_output.

Return type
climada.engine.unsequa.UncOutput

climada.engine.unsequa.calc_cost_benefit module

class climada.engine.unsequa.calc_cost_benefit.**CalcCostBenefit**(*haz_input_var:* InputVar | Hazard, *ent_input_var:* InputVar | Entity, *haz_fut_input_var:* InputVar | Hazard | *None* = *None*, *ent_fut_input_var:* InputVar | Entity | *None* = *None*)

Bases: *Calc*

Cost Benefit uncertainty analysis class

This is the base class to perform uncertainty analysis on the outputs of climada.engine.costbenefit.CostBenefit().

value_unit

Unit of the exposures value

Type
str

haz_input_var

Present Hazard uncertainty variable

Type
InputVar or *Hazard*

ent_input_var

Present Entity uncertainty variable

Type
InputVar or *Entity*

haz_unc_fut_Var

Future Hazard uncertainty variable

Type
InputVar or *Hazard*

ent_fut_input_var

Future Entity uncertainty variable

Type
InputVar or *Entity*

_input_var_names

Names of the required uncertainty variables ('haz_input_var', 'ent_input_var', 'haz_fut_input_var', 'ent_fut_input_var')

Type
tuple(str)

_metric_names

Names of the cost benefit output metrics ('tot_climate_risk', 'benefit', 'cost_ben_ratio', 'imp_meas_present', 'imp_meas_future')

Type
tuple(str)

__init__(*haz_input_var:* InputVar | Hazard, *ent_input_var:* InputVar | Entity, *haz_fut_input_var:* InputVar | Hazard | *None* = *None*, *ent_fut_input_var:* InputVar | Entity | *None* = *None*)

Initialize UncCalcCostBenefit

Sets the uncertainty input variables, the cost benefit metric_names, and the units.

Parameters

- **haz_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.hazard.Hazard Hazard uncertainty variable or Hazard for the present Hazard in climada.engine.CostBenefit.calc
- **ent_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.entity.Entity Entity uncertainty variable or Entity for the present Entity in climada.engine.CostBenefit.calc
- **haz_fut_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.hazard.Hazard, optional Hazard uncertainty variable or Hazard for the future Hazard The Default is None.
- **ent_fut_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.entity.Entity, optional Entity uncertainty variable or Entity for the future Entity in climada.engine.CostBenefit.calc

uncertainty(*unc_sample*, *processes=1*, *chunksize=None*, ***cost_benefit_kwargs*)

Computes the cost benefit for each sample in unc_output.sample_df.

By default, imp_meas_present, imp_meas_future, tot_climate_risk, benefit, cost_ben_ratio are computed.

This sets the attributes: unc_output.imp_meas_present_unc_df, unc_output.imp_meas_future_unc_df unc_output.tot_climate_risk_unc_df unc_output.benefit_unc_df unc_output.cost_ben_ratio_unc_df unc_output.unit unc_output.cost_benefit_kwargs

Parameters

- **unc_sample** (*climada.engine.uncertainty.unc_output.UncOutput*) – Uncertainty data object with the input parameters samples
- **processes** (*int, optional*) – Number of CPUs to use for parralel computations. The default is 1 (not parallel)
- **cost_benefit_kwargs** (*keyword arguments*) – Keyword arguments passed on to climada.engine.CostBenefit.calc()
- **chunksize** (*int, optional*) – Size of the sample chunks for parallel processing. Default is equal to the number of samples divided by the number of processes.

Returns
unc_output – Uncertainty data object in with the cost benefit outputs for each sample and all the sample data copied over from unc_sample.

Return type
climada.engine.uncertainty.unc_output.UncCostBenefitOutput

Raises

ValueError: – If no sampling parameters defined, the uncertainty distribution cannot be computed.

Notes

Parallelization logic is described in the base class here `Calc`

See also

climada.engine.cost_benefit
compute risk and adptation option cost benefits.

climada.engine.unsequa.calc_impact module

class climada.engine.unsequa.calc_impact.**CalcImpact**(*exp_input_var:* InputVar | Exposures, *impf_input_var:* InputVar | ImpactFuncSet, *haz_input_var:* InputVar | Hazard)

Bases: `Calc`

Impact uncertainty caclulation class.

This is the class to perform uncertainty analysis on the outputs of a climada.engine.impact.Impact() object.

rp

List of the chosen return periods.

Type
list(int)

calc_eai_exp

Compute eai_exp or not

Type
bool

calc_at_event

Compute eai_exp or not

Type
bool

value_unit

Unit of the exposures value

Type
str

exp_input_var

Exposure uncertainty variable

Type
InputVar or *Exposures*

impf_input_var

Impact function set uncertainty variable

Type
InputVar if ImpactFuncSet

haz_input_var

Hazard uncertainty variable

Type
InputVar or *Hazard*

_input_var_names

Names of the required uncertainty input variables ('exp_input_var', 'impf_input_var', 'haz_input_var')

Type
tuple(str)

_metric_names

Names of the impact output metrics ('aai_agg', 'freq_curve', 'at_event', 'eai_exp')

Type
tuple(str)

__init__(*exp_input_var:* InputVar | Exposures, *impf_input_var:* InputVar | ImpactFuncSet, *haz_input_var:* InputVar | Hazard)

Initialize UncCalcImpact

Sets the uncertainty input variables, the impact metric_names, and the units.

Parameters

- **exp_input_var** (*climada.engine.uncertainty.input_var.InputVar or climada.entity.Exposure*) – Exposure uncertainty variable or Exposure
- **impf_input_var** (*climada.engine.uncertainty.input_var.InputVar or climada.entity.ImpactFuncSet*) – Impact function set uncertainty variable or Impact function set
- **haz_input_var** (*climada.engine.uncertainty.input_var.InputVar or climada.hazard.Hazard*) – Hazard uncertainty variable or Hazard

uncertainty(*unc_sample*, *rp=None*, *calc_eai_exp=False*, *calc_at_event=False*, *processes=1*, *chunksize=None*)

Computes the impact for each sample in unc_data.sample_df.

By default, the aggregated average impact within a period of 1/frequency_unit (impact.aai_agg) and the excees impact at return periods rp (imppact.calc_freq_curve(self.rp).impact) is computed. Optionally, eai_exp and at_event is computed (this may require a larger amount of memory if the number of samples and/or the number of centroids and/or exposures points is large).

This sets the attributes self.rp, self.calc_eai_exp, self.calc_at_event, self.metrics.

This sets the attributes: unc_output.aai_agg_unc_df, unc_output.freq_curve_unc_df unc_output.eai_exp_unc_df unc_output.at_event_unc_df unc_output.unit

Parameters

- **unc_sample** (*climada.engine.uncertainty.unc_output.UncOutput*) – Uncertainty data object with the input parameters samples
- **rp** (*list(int), optional*) – Return periods in years to be computed. The default is [5, 10, 20, 50, 100, 250].
- **calc_eai_exp** (*boolean, optional*) – Toggle computation of the impact at each centroid location. The default is False.

- **calc_at_event** (*boolean, optional*) – Toggle computation of the impact for each event. The default is False.
- **processes** (*int, optional*) – Number of CPUs to use for parralel computations. The default is 1 (not parallel)
- **chunksize** (*int, optional*) – Size of the sample chunks for parallel processing. Default is equal to the number of samples divided by the number of processes.

Returns
unc_output – Uncertainty data object with the impact outputs for each sample and all the sample data copied over from unc_sample.

Return type
climada.engine.uncertainty.unc_output.UncImpactOutput

Raises
ValueError: – If no sampling parameters defined, the distribution cannot be computed.

Notes

Parallelization logic is described in the base class here `Calc`

See also

climada.engine.impact
compute impact and risk.

climada.engine.unsequa.input_var module

class climada.engine.unsequa.input_var.**InputVar**(*func: callable, distr_dict: Dict*)

Bases: `object`

Input variable for the uncertainty analysis

An uncertainty input variable requires a single or multi-parameter function. The parameters must follow a given distribution. The uncertainty input variables are the input parameters of the model.

distr_dict

Distribution of the uncertainty parameters. Keys are uncertainty parameters names and Values are probability density distribution from the scipy.stats package https://docs.scipy.org/doc/scipy/reference/stats.html

Type
dict

labels

Names of the uncertainty parameters (keys of distr_dict)

Type
list

func

User defined python fucntion with the uncertainty parameters as keyword arguements and which returns a climada object.

Type
function

Notes

A few default Variables are defined for Hazards, Exposures, Impact Fucntions, Measures and Entities.

Examples

Categorical variable function: LitPop exposures with m,n exponents in [0,5]

```
>>> import scipy as sp
>>> def litpop_cat(m, n):
...     exp = Litpop.from_countries('CHE', exponent=[m, n])
...     return exp
>>> distr_dict = {
...     'm': sp.stats.randint(low=0, high=5),
...     'n': sp.stats.randint(low=0, high=5)
...     }
>>> iv_cat = InputVar(func=litpop_cat, distr_dict=distr_dict)
```

Continuous variable function: Impact function for TC

```
>>> import scipy as sp
>>> def imp_fun_tc(G, v_half, vmin, k, _id=1):
...     intensity = np.linspace(0, 150, num=100)
...     mdd = np.repeat(1, len(intensity))
...     paa = np.array([sigmoid_function(v, G, v_half, vmin, k)
...                     for v in intensity])
...     imp_fun = ImpactFunc(haz_type='TC',
...                          id=_id,
...                          intensity_unit='m/s',
...                          intensity=intensity,
...                          mdd=mdd,
...                          paa=paa)
...     imp_fun.check()
...     impf_set = ImpactFuncSet([imp_fun])
...     return impf_set
>>> distr_dict = {"G": sp.stats.uniform(0.8, 1),
...       "v_half": sp.stats.uniform(50, 100),
...       "vmin": sp.stats.norm(loc=15, scale=30),
...       "k": sp.stats.randint(low=1, high=9)
...       }
>>> iv_cont = InputVar(func=imp_fun_tc, distr_dict=distr_dict)
```

__init__(*func: callable*, *distr_dict: Dict*)

Initialize InputVar

Parameters

- **func** (*function*) – Variable defined as a function of the uncertainty parameters
- **distr_dict** (*dict*) – Dictionary of the probability density distributions of the uncertainty parameters, with keys matching the keyword arguments (i.e. uncertainty parameters) of the func function. The distribution must be of type scipy.stats https://docs.scipy.org/doc/scipy/reference/stats.html

evaluate(***params*)

Return the value of uncertainty input variable.

By default, the value of the average is returned.

Parameters
****params** (*optional*) – Input parameters will be passed to self.func.

Returns
unc_func(params)** – Output of the uncertainty variable.

Return type
climada object

plot (*figsize=None*)

Plot the distributions of the parameters of the uncertainty variable.

Parameters
figsize (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

```
>>> nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
>>> figsize = (ncols * FIG_W, nrows * FIG_H)
```

Returns
axes – The figure and axes handle of the plot.

Return type
matplotlib.pyplot.figure, matplotlib.pyplot.axes

static var_to_inputvar (*var*)

Returns an uncertainty variable with no distribution if var is not an InputVar. Else, returns var.

Parameters
var (*climada.uncertainty.InputVar or any other CLIMADA object*)

Returns
var if var is InputVar, else InputVar with var and no distribution.

Return type
InputVar

static haz (*haz_list, n_ev=None, bounds_int=None, bounds_frac=None, bounds_freq=None*)

Helper wrapper for basic hazard uncertainty input variable

The following types of uncertainties can be added:

HE: sub-sampling events from the total event set
For each sub-sample, n_ev events are sampled with replacement. HE is the value of the seed for the uniform random number generator.

HI: scale the intensity of all events (homogeneously)
The instensity of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_int

HA: scale the fraction of all events (homogeneously)
The fraction of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_frac

HF: scale the frequency of all events (homogeneously)
The frequency of all events is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_freq

HL: sample uniformly from hazard list
From the provided list of hazard is elements are uniformly sampled. For example, Hazards outputs from dynamical models for different input factors.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **haz** (*List of climada.hazard.Hazard*) – The list of base hazard. Can be one or many to uniformly sample from.
- **n_ev** (*int, optional*) – Number of events to be subsampled per sample. Can be equal or larger than haz.size. The default is None.
- **bounds_int** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous intensity scaling. The default is None.
- **bounds_frac** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous fraction scaling. The default is None.
- **bounds_freq** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous frequency scaling. The default is None.

Returns
Uncertainty input variable for a hazard object.

Return type
climada.engine.unsequa.input_var.InputVar

static exp(*exp_list*, *bounds_totval=None*, *bounds_noise=None*)

Helper wrapper for basic exposure uncertainty input variable

The following types of uncertainties can be added:

ET: scale the total value (homogeneously)
The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_totvalue

EN: mutliplicative noise (inhomogeneous)
The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

EL: sample uniformly from exposure list
From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **exp_list** (*list of climada.entity.exposures.Exposures*) – The list of base exposure. Can be one or many to uniformly sample from.
- **bounds_totval** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous total value scaling.. The default is None.
- **bounds_noise** (*(float, float), optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.

Returns
Uncertainty input variable for an exposure object.

Return type
climada.engine.unsequa.input_var.InputVar

static impfset(*impf_set_list*, *haz_id_dict=None*, *bounds_mdd=None*, *bounds_paa=None*, *bounds_impfi=None*)

Helper wrapper for basic impact function set uncertainty input variable.

One impact function (chosen with haz_type and fun_id) is characterized.

The following types of uncertainties can be added:

MDD: scale the mdd (homogeneously)
The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

PAA: scale the paa (homogeneously)
The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

IFi: shift the intensity (homogeneously)
The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_int

IL: sample uniformly from impact function set list
From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **impf_set_list** (*list of ImpactFuncSet*) – The list of base impact function set. Can be one or many to uniformly sample from. The impact function ids must identical for all impact function sets.
- **bounds_mdd** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds_paa** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds_impfi** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.
- **haz_id_dict** (*dict(), optional*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int). Default is impsf_set.get_ids() i.e. all impact functions in the set

Returns
Uncertainty input variable for an impact function set object.

Return type
climada.engine.unsequa.input_var.InputVar

static ent(*impf_set_list*, *disc_rate*, *exp_list*, *meas_set*, *haz_id_dict*, *bounds_disc=None*, *bounds_cost=None*, *bounds_totval=None*, *bounds_noise=None*, *bounds_mdd=None*, *bounds_paa=None*, *bounds_impfi=None*)

Helper wrapper for basic entity set uncertainty input variable.

Important: only the impact function defined by haz_type and fun_id will be affected by bounds_impfi, bounds_mdd, bounds_paa.

The following types of uncertainties can be added:

DR: value of constant discount rate (homogeneously)
The value of the discounts in each year is sampled uniformly from a distribution with (min, max) = bounds_disc

CO: scale the cost (homogeneously)
The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = bounds_cost

ET: scale the total value (homogeneously)
The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_totval

EN: mutliplicative noise (inhomogeneous)
The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

EL: sample uniformly from exposure list
From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

MDD: scale the mdd (homogeneously)
The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

PAA: scale the paa (homogeneously)
The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

IFi: shift the intensity (homogeneously)
The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_int

IL: sample uniformly from impact function set list
From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **bounds_disk** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous discount rate scaling. The default is None.
- **bounds_cost** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous cost of all measures scaling. The default is None.
- **bounds_totval** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous total exposure value scaling. The default is None.
- **bounds_noise** (*(float, float), optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.
- **bounds_mdd** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds_paa** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds_impfi** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.

- **impf_set_list** (*list of ImpactFuncSet*) – The list of base impact function set. Can be one or many to uniformly sample from. The impact function ids must identical for all impact function sets.
- **disc_rate** (*climada.entity.disc_rates.base.DiscRates*) – The base discount rates.
- **exp_list** (*[climada.entity.exposures.base.Exposure]*) – The list of base exposure. Can be one or many to uniformly sample from.
- **meas_set** (*climada.entity.measures.measure_set.MeasureSet*) – The base measures.
- **haz_id_dict** (*dict*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int).

Returns
Entity uncertainty input variable

Return type
climada.engine.unsequa.input_var.InputVar

static entfut(*impf_set_list*, *exp_list*, *meas_set*, *haz_id_dict*, *bounds_cost=None*, *bounds_eg=None*, *bounds_noise=None*, *bounds_impfi=None*, *bounds_mdd=None*, *bounds_paa=None*)

Helper wrapper for basic future entity set uncertainty input variable.

Important: only the impact function defined by haz_type and fun_id will be affected by bounds_impfi, bounds_mdd, bounds_paa.

The following types of uncertainties can be added:

CO: scale the cost (homogeneously)
The cost of all measures is multiplied by the same number sampled uniformly from a distribution with (min, max) = bounds_cost

EG: scale the exposures growth (homogeneously)
The value at each exposure point is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_eg

EN: mutliplicative noise (inhomogeneous)
The value of each exposure point is independently multiplied by a random number sampled uniformly from a distribution with (min, max) = bounds_noise. EN is the value of the seed for the uniform random number generator.

EL: sample uniformly from exposure list
From the provided list of exposure is elements are uniformly sampled. For example, LitPop instances with different exponents.

MDD: scale the mdd (homogeneously)
The value of mdd at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_mdd

PAA: scale the paa (homogeneously)
The value of paa at each intensity is multiplied by a number sampled uniformly from a distribution with (min, max) = bounds_paa

IFi: shift the impact function intensity (homogeneously)
The value intensity are all summed with a random number sampled uniformly from a distribution with (min, max) = bounds_impfi

IL: sample uniformly from impact function set list
From the provided list of impact function sets elements are uniformly sampled. For example, impact functions obtained from different calibration methods.

If a bounds is None, this parameter is assumed to have no uncertainty.

Parameters

- **bounds_cost** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous cost of all measures scaling. The default is None.
- **bounds_eg** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous total exposure growth scaling. The default is None.
- **bounds_noise** (*(float, float), optional*) – Bounds of the uniform distribution to scale each exposure point independently. The default is None.
- **bounds_mdd** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous mdd scaling. The default is None.
- **bounds_paa** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous paa scaling. The default is None.
- **bounds_impfi** (*(float, float), optional*) – Bounds of the uniform distribution for the homogeneous shift of intensity. The default is None.
- **impf_set_list** (*list of ImpactFuncSet*) – The list of base impact function set. Can be one or many to uniformly sample from. The impact function ids must identical for all impact function sets.
- **exp_list** (*[climada.entity.exposures.base.Exposure]*) – The list of base exposure. Can be one or many to uniformly sample from.
- **meas_set** (*climada.entity.measures.measure_set.MeasureSet*) – The base measures.
- **haz_id_dict** (*dict*) – Dictionary of the impact functions affected by uncertainty. Keys are hazard types (str), values are a list of impact function id (int).

Returns
Entity uncertainty input variable

Return type
climada.engine.unsequa.input_var.InputVar

climada.engine.unsequa.unc_output module

class climada.engine.unsequa.unc_output.**UncOutput**(*samples_df*, *unit=None*)

Bases: `object`

Class to store and plot uncertainty and sensitivity analysis output data

This is the base class to store uncertainty and sensitivity outputs of an analysis done on climada.engine.impact.Impact() or climada.engine.costbenefit.CostBenefit() object.

samples_df

Values of the sampled uncertainty parameters. It has n_samples rows and one column per uncertainty parameter.

Type
pandas.DataFrame

distr_dict

Comon flattened dictionary of all the distr_dict of all input variables. It represents the distribution of all the uncertainty parameters.

Type
dict

__init__(*samples_df*, *unit=None*)

Initialize Uncertainty Data object.

Parameters

- **samples_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str, optional*) – value unit

order_samples(*by_parameters*)

Function to sort the samples dataframe.

Note: the unc_output.samples_df is ordered inplace.

Parameters
by_parameters (*list[string]*) – List of the uncertainty parameters to sort by (ordering in list is kept)

Return type
None.

get_samples_df()

get_unc_df(*metric_name*)

set_unc_df(*metric_name*, *unc_df*)

get_sens_df(*metric_name*)

set_sens_df(*metric_name*, *sens_df*)

check_salib(*sensitivity_method*)

Checks whether the chosen sensitivity method and the sampling method used to generated self.samples_df respect the pairing recommendation by the SALib package.

https://salib.readthedocs.io/en/latest/api.html

Parameters
sensitivity_method (*str*) – Name of the sensitivity analysis method.

Returns
True if sampling and sensitivity methods respect the recommended pairing.

Return type
bool

property sampling_method

Returns the sampling method used to generate self.samples_df See: https://salib.readthedocs.io/en/latest/api.html#

Returns
Sampling method name

Return type
str

property sampling_kwargs

Returns the kwargs of the sampling method that generate self.samples_df

Returns
Dictionary of arguments for SALib sampling method

Return type
dict

property n_samples

The effective number of samples

Returns
effective number of samples

Return type
int

property param_labels

Labels of all uncertainty input parameters.

Returns
Labels of all uncertainty input parameters.

Return type
list of str

property problem_sa

The description of the uncertainty variables and their distribution as used in SALib. https://salib.readthedocs.io/en/latest/basics.html

Returns
Salib problem dictionary.

Return type
dict

property uncertainty_metrics

Retrieve all uncertainty output metrics names

Returns
unc_metric_list – List of names of attributes containing metrics uncertainty values, without the trailing '_unc_df'

Return type
[str]

property sensitivity_metrics

Retrieve all sensitivity output metrics names

Returns
sens_metric_list – List of names of attributes containing metrics sensitivity values, without the trailing '_sens_df'

Return type
[str]

get_uncertainty(*metric_list=None*)

Returns uncertainty dataframe with values for each sample

Parameters
metric_list (*[str], optional*) – List of uncertainty metrics to consider. The default returns all uncertainty metrics at once.

Returns
Joint dataframe of all uncertainty values for all metrics in the metric_list.

Return type
pandas.DataFrame

See also

uncertainty_metrics
list of all available uncertainty metrics

get_sensitivity(*salib_si*, *metric_list=None*)

Returns sensitivity index

E.g. For the sensitivity analysis method 'sobol', the choices are ['S1', 'ST'], for 'delta' the choices are ['delta', 'S1'].

For more information see the SAlib documentation: https://salib.readthedocs.io/en/latest/basics.html

Parameters

- **salib_si** (*str*) – Sensitivity index
- **metric_list** (*[str], optional*) – List of sensitivity metrics to consider. The default returns all sensitivity indices at once.

Returns
Joint dataframe of the sensitvity indices for all metrics in the metric_list

Return type
pandas.DataFrame

See also

sensitvity_metrics
list of all available sensitivity metrics

get_largest_si(*salib_si*, *metric_list=None*, *threshold=0.01*)

Get largest si per metric

Parameters

- **salib_si** (*str*) – The name of the sensitivity index to plot.
- **metric_list** (*list of strings, optional*) – List of metrics to plot the sensitivity. Default is None.
- **threshold** (*float*) – The minimum value a sensitivity index must have to be considered as the largest. The default is 0.01.

Returns
max_si_df – Dataframe with the largest si and its value per metric

Return type
pandas.dataframe

plot_sample(*figsize=None*)

Plot the sample distributions of the uncertainty input parameters

For each uncertainty input variable, the sample distributions is shown in a separate axes.

Parameters

figsize (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

```
>>> nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
>>> figsize = (ncols * FIG_W, nrows * FIG_H)
```

Raises

ValueError – If no sample was computed the plot cannot be made.

Returns

axes – The axis handle of the plot.

Return type

matplotlib.pyplot.axes

plot_uncertainty(*metric_list=None*, *orig_list=None*, *figsize=None*, *log=False*, *axes=None*, *calc_delta=False*)

Plot the uncertainty distribution

For each risk metric, a separate axes is used to plot the uncertainty distribution of the output values obtained over the sampled input parameters.

Parameters

- **metric_list** (*list[str], optional*) – List of metrics to plot the distribution. The default is None.
- **orig_list** (*list[float], optional*) – List of the original (without uncertainty) values for each sub-metric of the mtrics in metric_list. The ordering is identical. The default is None.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as: nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3) figsize = (ncols * FIG_W, nrows * FIG_H)
- **log** (*boolean, optional*) – Use log10 scale for x axis. Default is False.
- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **calc_delta** (*boolean, optional*) – Adapt x axis label for CalcDeltaImpact unc_output. Default is False.

Raises

ValueError – If no metric distribution was computed the plot cannot be made.

Returns

axes – The axes handle of the plot.

Return type

matplotlib.pyplot.axes

See also

uncertainty_metrics
list of all available uncertainty metrics

plot_rp_uncertainty(*orig_list=None*, *figsize=(16, 6)*, *axes=None*, *calc_delta=False*)

Plot the distribution of return period uncertainty

Parameters

- **orig_list** (*list[float], optional*) – List of the original (without uncertainty) values for each sub-metric of the metrics in metric_list. The ordering is identical. The default is None.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is (16, 6)
- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **calc_delta** (*boolean, optional*) – Adapt axis labels for CalcDeltaImpact unc_output. Default is False.

Raises
`ValueError` – If no metric distribution was computed the plot cannot be made.

Returns
axes – The axis handle of the plot.

Return type
matplotlib.pyplot.axes

`plot_sensitivity`(*salib_si='S1'*, *salib_si_conf='S1_conf'*, *metric_list=None*, *figsize=None*, *axes=None*, ***kwargs*)

Bar plot of a first order sensitivity index

For each metric, the sensitivity indices are plotted in a separate axes.

This requires that a senstivity analysis was already performed.

E.g. For the sensitivity analysis method 'sobol', the choices are ['S1', 'ST'], for 'delta' the choices are ['delta', 'S1'].

Note that not all sensitivity indices have a confidence interval.

For more information see the SAlib documentation: https://salib.readthedocs.io/en/latest/basics.html

Parameters

- **salib_si** (*string, optional*) – The first order (one value per metric output) sensitivity index to plot. The default is S1.
- **salib_si_conf** (*string, optional*) – The confidence value for the first order sensitivity index to plot. The default is S1_conf.
- **metric_list** (*list of strings, optional*) – List of metrics to plot the sensitivity. If a metric is not found it is ignored.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

```
>>> nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
>>> figsize = (ncols * FIG_W, nrows * FIG_H)
```

- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **kwargs** – Keyword arguments passed on to pandas.DataFrame.plot(kind='bar')

Raises
`ValueError :` – If no sensitivity is available the plot cannot be made.

Returns
axes – The axes handle of the plot.

Return type
matplotlib.pyplot.axes

See also

sensitvity_metrics
list of all available sensitivity metrics

plot_sensitivity_second_order(*salib_si='S2'*, *salib_si_conf='S2_conf'*, *metric_list=None*, *figsize=None*, *axes=None*, ***kwargs*)

Plot second order sensitivity indices as matrix.

For each metric, the sensitivity indices are plotted in a separate axes.

E.g. For the sensitivity analysis method 'sobol', the choices are ['S2', 'S2_conf'].

Note that not all sensitivity indices have a confidence interval.

For more information see the SAlib documentation: https://salib.readthedocs.io/en/latest/basics.html

Parameters

- **salib_si** (*string, optional*) – The second order sensitivity indexto plot. The default is S2.
- **salib_si_conf** (*string, optional*) – The confidence value for thesensitivity index salib_si to plot. The default is S2_conf.
- **metric_list** (*list of strings, optional*) – List of metrics to plot the sensitivity. If a metric is not found it is ignored. Default is all 1D metrics.
- **figsize** (*tuple(int or float, int or float), optional*) – The figsize argument of matplotlib.pyplot.subplots() The default is derived from the total number of plots (nplots) as:

```
>>> nrows, ncols = int(np.ceil(nplots / 3)), min(nplots, 3)
>>> figsize = (ncols * 5, nrows * 5)
```

- **axes** (*matplotlib.pyplot.axes, optional*) – Axes handles to use for the plot. The default is None.
- **kwargs** – Keyword arguments passed on to matplotlib.pyplot.imshow()

Raises
ValueError : – If no sensitivity is available the plot cannot be made.

Returns
axes – The axes handle of the plot.

Return type
matplotlib.pyplot.axes

See also

sensitvity_metrics
list of all available sensitivity metrics

plot_sensitivity_map(*salib_si='S1'*, ***kwargs*)

Plot a map of the largest sensitivity index in each exposure point

Requires the uncertainty distribution for eai_exp.

Parameters

- **salib_si** (*str, optional*) – The name of the sensitivity index to plot. The default is 'S1'.
- **kwargs** – Keyword arguments passed on to climada.util.plot.geo_scatter_categorical

Raises
ValueError : – If no sensitivity data is found, raise error.

Returns
ax – The axis handle of the plot.

Return type
matplotlib.pyplot.axes

See also

climada.util.plot.geo_scatter_categorical
geographical plot for categorical variable

to_hdf5(*filename=None*)

Save output to .hdf5

Parameters
filename (*str or pathlib.Path, optional*) – The filename with absolute or relative path. The default name is "unc_output + datetime.now() + .hdf5" and the default path is taken from climada.config

Returns
save_path – Path to the saved file

Return type
pathlib.Path

static from_hdf5(*filename*)

Load a uncertainty and uncertainty output data from .hdf5 file

Parameters
filename (*str or pathlib.Path*) – The filename with absolute or relative path.

Returns
unc_output – Uncertainty and sensitivity data loaded from .hdf5 file.

Return type
climada.engine.uncertainty.unc_output.UncOutput

class climada.engine.unsequa.unc_output.**UncCostBenefitOutput**(*samples_df*, *unit*, *imp_meas_present_unc_df*, *imp_meas_future_unc_df*, *tot_climate_risk_unc_df*, *benefit_unc_df*, *cost_ben_ratio_unc_df*, *cost_benefit_kwargs*)

Bases: *UncOutput*

Extension of UncOutput specific for CalcCostBenefit, returned by the uncertainty() method.

__init__(*samples_df*, *unit*, *imp_meas_present_unc_df*, *imp_meas_future_unc_df*, *tot_climate_risk_unc_df*, *benefit_unc_df*, *cost_ben_ratio_unc_df*, *cost_benefit_kwargs*)

Constructor

Uncertainty output values from cost_benefit.calc for each sample

Parameters

- **samples_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str*) – value unit
- **imp_meas_present_unc_df** (*pandas.DataFrame*) – Each row contains the values of imp_meas_present for one sample (row of samples_df)
- **imp_meas_future_unc_df** (*pandas.DataFrame*) – Each row contains the values of imp_meas_future for one sample (row of samples_df)
- **tot_climate_risk_unc_df** (*pandas.DataFrame*) – Each row contains the values of tot_cliamte_risk for one sample (row of samples_df)
- **benefit_unc_df** (*pandas.DataFrame*) – Each row contains the values of benefit for one sample (row of samples_df)
- **cost_ben_ratio_unc_df** (*pandas.DataFrame*) – Each row contains the values of cost_ben_ratio for one sample (row of samples_df)
- **cost_benefit_kwargs** (*pandas.DataFrame*) – Each row contains the value of cost_benefit for one sample (row of samples_df)

class climada.engine.unsequa.unc_output.**UncImpactOutput**(*samples_df*, *unit*, *aai_agg_unc_df*, *freq_curve_unc_df*, *eai_exp_unc_df*, *at_event_unc_df*, *coord_df*)

Bases: *UncOutput*

Extension of UncOutput specific for CalcImpact, returned by the uncertainty() method.

__init__(*samples_df*, *unit*, *aai_agg_unc_df*, *freq_curve_unc_df*, *eai_exp_unc_df*, *at_event_unc_df*, *coord_df*)

Constructor

Uncertainty output values from impact.calc for each sample

Parameters

- **samples_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str*) – value unit
- **aai_agg_unc_df** (*pandas.DataFrame*) – Each row contains the value of aai_aag for one sample (row of samples_df)
- **freq_curve_unc_df** (*pandas.DataFrame*) – Each row contains the values of the impact exceedence frequency curve for one sample (row of samples_df)
- **eai_exp_unc_df** (*pandas.DataFrame*) – Each row contains the values of eai_exp for one sample (row of samples_df)
- **at_event_unc_df** (*pandas.DataFrame*) – Each row contains the values of at_event for one sample (row of samples_df)
- **coord_df** (*pandas.DataFrame*) – Coordinates of the exposure

class climada.engine.unsequa.unc_output.**UncDeltaImpactOutput**(*samples_df*, *unit*, *aai_agg_unc_df*, *freq_curve_unc_df*, *eai_exp_unc_df*, *at_event_initial_unc_df*, *at_event_final_unc_df*, *coord_df*)

Bases: `UncOutput`

Extension of UncOutput specific for CalcDeltaImpact, returned by the uncertainty() method.

__init__(*samples_df*, *unit*, *aai_agg_unc_df*, *freq_curve_unc_df*, *eai_exp_unc_df*, *at_event_initial_unc_df*, *at_event_final_unc_df*, *coord_df*)

Constructor

Uncertainty output values from impact.calc for each sample

Parameters

- **samples_df** (*pandas.DataFrame*) – input parameters samples
- **unit** (*str*) – value unit
- **aai_agg_unc_df** (*pandas.DataFrame*) – Each row contains the value of aai_aag for one sample (row of samples_df)
- **freq_curve_unc_df** (*pandas.DataFrame*) – Each row contains the values of the impact exceedence frequency curve for one sample (row of samples_df)
- **eai_exp_unc_df** (*pandas.DataFrame*) – Each row contains the values of eai_exp for one sample (row of samples_df)
- **at_event_initial_unc_df** (*pandas.DataFrame*) – Each row contains the values of at_event for one sample (row of samples_df)
- **at_event_final_unc_df** (*pandas.DataFrame*) – Each row contains the values of at_event for one sample (row of samples_df)
- **coord_df** (*pandas.DataFrame*) – Coordinates of the exposure

climada.engine.unsequa.calc_delta_climate module

class climada.engine.unsequa.calc_delta_climate.**CalcDeltaImpact**(*exp_initial_input_var:* InputVar | Exposures, *impf_initial_input_var:* InputVar | ImpactFuncSet, *haz_initial_input_var:* InputVar | Hazard, *exp_final_input_var:* InputVar | Exposures, *impf_final_input_var:* InputVar | ImpactFuncSet, *haz_final_input_var:* InputVar | Hazard)

Bases: `Calc`

Delta Impact uncertainty caclulation class.

This is the class to perform uncertainty analysis on the outputs of a relative change in impact from a (final impact - initial impact) / initial impact. Impact objects are regular climada.engine.impact.Impact() objects. The resulting Delta Impact is a relative change (fraction of the inital impact). The relative change is intuitive to understand in contrast to absolute changes which are hard to understand without knowledge of the absolute initial (baseline) state.

rp

List of the chosen return periods.

Type
list(int)

calc_eai_exp

Compute eai_exp or not

Type
bool

calc_at_event

Compute eai_exp or not

Type
bool

value_unit

Unit of the exposures value

Type
str

exp_input_var

Exposure uncertainty variable

Type
InputVar or *Exposures*

impf_input_var

Impact function set uncertainty variable

Type
InputVar if ImpactFuncSet

haz_input_var

Hazard uncertainty variable

Type
InputVar or *Hazard*

_input_var_names

Names of the required uncertainty input variables ('exp_initial_input_var', 'impf_initial_input_var', 'haz_initial_input_var', 'exp_final_input_var', 'impf_final_input_var', 'haz_final_input_var")

Type
tuple(str)

_metric_names

Names of the impact output metrics ('aai_agg', 'freq_curve', 'at_event', 'eai_exp')

Type
tuple(str)

__init__(*exp_initial_input_var:* InputVar | Exposures, *impf_initial_input_var:* InputVar | ImpactFuncSet, *haz_initial_input_var:* InputVar | Hazard, *exp_final_input_var:* InputVar | Exposures, *impf_final_input_var:* InputVar | ImpactFuncSet, *haz_final_input_var:* InputVar | Hazard)

Initialize UncCalcImpact

Sets the uncertainty input variables, the impact metric_names, and the units.

Parameters

- **exp_initial_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.entity.Exposure Exposure uncertainty variable or Exposure of initial state
- **impf_initital_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.entity.ImpactFuncSet Impact function set uncertainty variable or Impact function set of initial state
- **haz_initial_input_var** (*climada.engine.uncertainty.input_var.InputVar*) – or climada.hazard.Hazard Hazard uncertainty variable or Hazard of initial state
- **exp_final_input_var** (*climada.engine.uncertainty.input_var.InputVar or*) – climada.entity.Exposure Exposure uncertainty variable or Exposure of final state
- **impf_final_input_var** (*climada.engine.uncertainty.input_var.InputVar or*) – climada.entity.ImpactFuncSet Impact function set uncertainty variable or Impact function set of final state
- **haz_final_input_var** (*climada.engine.uncertainty.input_var.InputVar or*) – climada.hazard.Hazard Hazard uncertainty variable or Hazard of final state

uncertainty(*unc_sample*, *rp=None*, *calc_eai_exp=False*, *calc_at_event=False*, *relative_delta=True*, *processes=1*, *chunksize=None*)

Computes the differential impact between the reference (initial) and future (final) for each sample in unc_data.sample_df.

By default, the aggregated average impact within a period of 1/frequency_unit (impact.aai_agg) and the excess impact at return periods rp (imppact.calc_freq_curve(self.rp).impact) is computed. Optionally, eai_exp and at_event is computed (this may require a larger amount of memory if the number of samples and/or the number of centroids and/or exposures points is large). For all metrics, the impacts are caculated first and the the difference thereof is computed. For example: (impact_final.aai_agg - impact_inital.aai_agg / impact_inital.aai_agg)

This sets the attributes self.rp, self.calc_eai_exp, self.calc_at_event, self.metrics.

This sets the attributes: unc_output.aai_agg_unc_df, unc_output.freq_curve_unc_df unc_output.eai_exp_unc_df unc_output.at_event_unc_df unc_output.unit

Parameters

- **unc_sample** (*climada.engine.uncertainty.unc_output.UncOutput*) – Uncertainty data object with the input parameters samples
- **rp** (*list(int), optional*) – Return periods in years to be computed. The default is [5, 10, 20, 50, 100, 250].
- **calc_eai_exp** (*boolean, optional*) – Toggle computation of the impact at each centroid location. The default is False.
- **calc_at_event** (*boolean, optional*) – Toggle computation of the impact for each event. The default is False.
- **relative_delta** (*bool, optional*) – Normalize delta impacts by past impacts or not. The default is True.
- **processes** (*int, optional*) – Number of CPUs to use for parralel computations. The default is 1 (not parallel)
- **chunksize** (*int, optional*) – Size of the sample chunks for parallel processing. Default is equal to the number of samples divided by the number of processes.

Returns

unc_output – Uncertainty data object with the delta impact outputs for each sample and all the sample data copied over from unc_sample.

Return type
climada.engine.uncertainty.unc_output.UncImpactOutput

Raises
ValueError: – If no sampling parameters defined, the distribution cannot be computed.

Notes

Parallelization logic is described in the base class here `Calc`

See also

climada.engine.impact
compute impact and risk.

climada.engine.calibration_opt module

climada.engine.calibration_opt.**calib_instance**(*hazard*, *exposure*, *impact_func*, *df_out=Empty DataFrame Columns: [] Index: []*, *yearly_impact=False*, *return_cost='False'*)

calculate one impact instance for the calibration algorithm and write to given DataFrame

Parameters

- **hazard** (*Hazard*)
- **exposure** (*Exposure*)
- **impact_func** (*ImpactFunc*)
- **df_out** (*Dataframe, optional*) – Output DataFrame with headers of columns defined and optionally with first row (index=0) defined with values. If columns "impact", "event_id", or "year" are not included, they are created here. Data like reported impacts or impact function parameters can be given here; values are preserved.
- **yearly_impact** (*boolean, optional*) – if set True, impact is returned per year, not per event
- **return_cost** (*str, optional*) – if not 'False' but any of 'R2', 'logR2', cost is returned instead of df_out

Returns
df_out – DataFrame with modelled impact written to rows for each year or event.

Return type
DataFrame

climada.engine.calibration_opt.**init_impf**(*impf_name_or_instance, param_dict, df_out=Empty DataFrame Columns: [] Index: [0]*)

create an ImpactFunc based on the parameters in param_dict using the method specified in impf_parameterisation_name and document it in df_out.

Parameters

- **impf_name_or_instance** (*str or ImpactFunc*) – method of impact function parameterisation e.g. 'emanuel' or an instance of ImpactFunc
- **param_dict** (*dict, optional*) – dict of parameter_names and values e.g. {'v_thresh': 25.7, 'v_half': 70, 'scale': 1} or {'mdd_shift': 1.05, 'mdd_scale': 0.8, 'paa_shift': 1, paa_scale': 1}

Returns

- **imp_fun** (*ImpactFunc*) – The Impact function based on the parameterisation
- **df_out** (*DataFrame*) – Output DataFrame with headers of columns defined and with first row (index=0) defined with values. The impact function parameters from param_dict are represented here.

climada.engine.calibration_opt.**change_impf**(*impf_instance*, *param_dict*)

apply a shifting or a scaling defined in param_dict to the impact function in impf_istance and return it as a new ImpactFunc object.

Parameters

- **impf_instance** (*ImpactFunc*) – an instance of ImpactFunc
- **param_dict** (*dict*) – dict of parameter_names and values (interpreted as factors, 1 = neutral) e.g. {'mdd_shift': 1.05, 'mdd_scale': 0.8, 'paa_shift': 1, paa_scale': 1}

Returns
ImpactFunc

Return type
The Impact function based on the parameterisation

climada.engine.calibration_opt.**init_impact_data**(*hazard_type*, *region_ids*, *year_range*, *source_file*, *reference_year*, *impact_data_source='emdat'*, *yearly_impact=True*)

creates a dataframe containing the recorded impact data for one hazard type and one area (countries, country or local split)

Parameters

- **hazard_type** (*str*) – default = 'TC', type of hazard 'WS','FL' etc.
- **region_ids** (*str*) – name the region_ids or country names
- **year_range** (*list*) – list containting start and end year. e.g. [1980, 2017]
- **source_file** (*str*)
- **reference_year** (*int*) – impacts will be scaled to this year
- **impact_data_source** (*str, optional*) – default 'emdat', others maybe possible
- **yearly_impact** (*bool, optional*) – if set True, impact is returned per year, not per event

Returns
df_out – Dataframe with recorded impact written to rows for each year or event.

Return type
pd.DataFrame

climada.engine.calibration_opt.**calib_cost_calc**(*df_out*, *cost_function*)

calculate the cost function of the modelled impact impact_CLIMADA and
the reported impact impact_scaled in df_out

Parameters

- **df_out** (*pd.Dataframe*) – DataFrame as created in calib_instance
- **cost_function** (*str*) – chooses the cost function e.g. 'R2' or 'logR2'

Returns
cost – The results of the cost function when comparing modelled and reported impact

Return type
float

climada.engine.calibration_opt.**calib_all**(*hazard*, *exposure*, *impf_name_or_instance*, *param_full_dict*, *impact_data_source*, *year_range*, *yearly_impact=True*)

portrait the difference between modelled and reported impacts for all impact functions described in param_full_dict and impf_name_or_instance

Parameters

- **hazard** (*list or Hazard*)
- **exposure** (*list or Exposures*) – list or instance of exposure of full countries
- **impf_name_or_instance** (*string or ImpactFunc*) – the name of a parameterisation or an instance of class ImpactFunc e.g. 'emanuel'
- **param_full_dict** (*dict*) – a dict containing keys used for f_name_or_instance and values which are iterable (lists) e.g. {'v_thresh' : [25.7, 20], 'v_half': [70], 'scale': [1, 0.8]}
- **impact_data_source** (*dict or pd.Dataframe*) – with name of impact data source and file location or dataframe
- **year_range** (*list*)
- **yearly_impact** (*bool, optional*)

Returns
df_result – df with modelled impact written to rows for each year or event.

Return type
pd.DataFrame

climada.engine.calibration_opt.**calib_optimize**(*hazard*, *exposure*, *impf_name_or_instance*, *param_dict*, *impact_data_source*, *year_range*, *yearly_impact=True*, *cost_fucntion='R2'*, *show_details=False*)

portrait the difference between modelled and reported impacts for all impact functions described in param_full_dict and impf_name_or_instance

Parameters

- **hazard** (*list or Hazard*)
- **exposure** (*list or Exposures*) – list or instance of exposure of full countries
- **impf_name_or_instance** (*string or ImpactFunc*) – the name of a parameterisation or an instance of class ImpactFunc e.g. 'emanuel'
- **param_dict** (*dict*) – a dict containing keys used for impf_name_or_instance and one set of values e.g. {'v_thresh': 25.7, 'v_half': 70, 'scale': 1}
- **impact_data_source** (*dict or pd. dataframe*) – with name of impact data source and file location or dataframe
- **year_range** (*list*)
- **yearly_impact** (*bool, optional*)
- **cost_function** (*str, optional*) – the argument for function calib_cost_calc, default 'R2'
- **show_details** (*bool, optional*) – if True, return a tuple with the parameters AND the details of the optimization like success, status, number of iterations etc

Returns
param_dict_result – the parameters with the best calibration results (or a tuple with (1) the parameters and (2) the optimization output)

Return type
dict or tuple

climada.engine.cost_benefit module

class climada.engine.cost_benefit.**CostBenefit**(*present_year: int = 2016, future_year: int = 2030, tot_climate_risk: float = 0.0, unit: str = 'USD', color_rgb: Dict[str, ndarray] | None = None, benefit: Dict[str, float] | None = None, cost_ben_ratio: Dict[str, float] | None = None, imp_meas_present: Dict[str, float | Tuple[float, float] | Impact | ImpactFreqCurve] | None = None, imp_meas_future: Dict[str, float | Tuple[float, float] | Impact | ImpactFreqCurve] | None = None*)

Bases: object

Impact definition. Compute from an entity (exposures and impact functions) and hazard.

present_year

present reference year

Type
int

future_year

future year

Type
int

tot_climate_risk

total climate risk without measures

Type
float

unit

unit used for impact

Type
str

color_rgb

color code RGB for each measure.

Type
dict

Key

measure name ('no measure' used for case without measure),

Type
str

Value

Type
np.array

benefit

benefit of each measure. Key: measure name, Value: float benefit

Type
dict

cost_ben_ratio

cost benefit ratio of each measure. Key: measure name, Value: float cost benefit ratio

Type
dict

imp_meas_future

impact of each measure at future or default. Key: measure name ('no measure' used for case without measure), Value: dict with: 'cost' (tuple): (cost measure, cost factor insurance), 'risk' (float): risk measurement, 'risk_transf' (float): annual expected risk transfer, 'efc' (ImpactFreqCurve): impact exceedance freq (optional) 'impact' (Impact): impact instance

Type
dict

imp_meas_present

impact of each measure at present. Key: measure name ('no measure' used for case without measure), Value: dict with: 'cost' (tuple): (cost measure, cost factor insurance), 'risk' (float): risk measurement, 'risk_transf' (float): annual expected risk transfer, 'efc' (ImpactFreqCurve): impact exceedance freq (optional) 'impact' (Impact): impact instance

Type
dict

__init__(*present_year: int = 2016*, *future_year: int = 2030*, *tot_climate_risk: float = 0.0*, *unit: str = 'USD'*, *color_rgb: Dict[str, ndarray] | None = None*, *benefit: Dict[str, float] | None = None*, *cost_ben_ratio: Dict[str, float] | None = None*, *imp_meas_present: Dict[str, float | Tuple[float, float] |* Impact *|* ImpactFreqCurve*] | None = None*, *imp_meas_future: Dict[str, float | Tuple[float, float] |* Impact *|* ImpactFreqCurve*] | None = None*)

Initilization

calc(*hazard*, *entity*, *haz_future=None*, *ent_future=None*, *future_year=None*, *risk_func=<function risk_aai_agg>*, *imp_time_depen=None*, *save_imp=False*, *assign_centroids=True*)

Compute cost-benefit ratio for every measure provided current and, optionally, future conditions. Present and future measures need to have the same name. The measures costs need to be discounted by the user. If future entity provided, only the costs of the measures of the future and the discount rates of the present will be used.

Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.entity*)
- **haz_future** (*climada.Hazard, optional*) – hazard in the future (future year provided at ent_future)
- **ent_future** (*Entity, optional*) – entity in the future. Default is None
- **future_year** (*int, optional*) – future year to consider if no ent_future. Default is None provided. The benefits are added from the entity.exposures.ref_year until ent_future.exposures.ref_year, or until future_year if no ent_future given. Default: entity.exposures.ref_year+1

- **risk_func** (*func optional*) – function describing risk measure to use to compute the annual benefit from the Impact. Default: average annual impact (aggregated).
- **imp_time_depen** (*float, optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.
- **save_imp** (*bool, optional*) – Default: False
- **assign_centroids** (*bool, optional*) – indicates whether centroids are assigned to the self.exposures object. Centroids assignment is an expensive operation; set this to `False` to save computation time if the exposures from `ent` and `ent_fut` have already centroids assigned for the respective hazards. Default: True
- **True if Impact of each measure is saved. Default is False.**

combine_measures(*in_meas_names*, *new_name*, *new_color*, *disc_rates*, *imp_time_depen=None*, *risk_func=<function risk_aai_agg>*)

Compute cost-benefit of the combination of measures previously computed by calc with save_imp=True. The benefits of the measures per event are added. To combine with risk transfer options use apply_risk_transfer.

Parameters

- **in_meas_names** (*list(str)*)
- **list with names of measures to combine**
- **new_name** (*str*) – name to give to the new resulting measure new_color (np.array): color code RGB for new measure, e.g. np.array([0.1, 0.1, 0.1])
- **disc_rates** (*DiscRates*) – discount rates instance
- **imp_time_depen** (*float, optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.
- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).

Return type
climada.CostBenefit

apply_risk_transfer(*meas_name*, *attachment*, *cover*, *disc_rates*, *cost_fix=0*, *cost_factor=1*, *imp_time_depen=None*, *risk_func=<function risk_aai_agg>*)

Applies risk transfer to given measure computed before with saved impact and compares it to when no measure is applied. Appended to dictionaries of measures.

Parameters

- **meas_name** (*str*) – name of measure where to apply risk transfer
- **attachment** (*float*) – risk transfer values attachment (deductible)
- **cover** (*float*) – risk transfer cover
- **cost_fix** (*float*) – fixed cost of implemented innsurance, e.g. transaction costs
- **cost_factor** (*float, optional*) – factor to which to multiply the insurance layer to compute its cost. Default is 1
- **imp_time_depen** (*float, optional*) – parameter which represents time evolution of impact (super- or sublinear). If None: all years count the same when there is no future hazard nor entity and 1 (linear annual change) when there is future hazard or entity. Default is None.

- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).

remove_measure(*meas_name*)

Remove computed values of given measure

Parameters

meas_name (*str*) – name of measure to remove

plot_cost_benefit(*cb_list=None*, *axis=None*, ***kwargs*)

Plot cost-benefit graph. Call after calc().

Parameters

- **cb_list** (*list(CostBenefit), optional*) – if other CostBenefit provided, overlay them all. Used for uncertainty visualization.
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for Rectangle matplotlib, e.g. alpha=0.5 (color is set by measures color attribute)

Return type

matplotlib.axes._subplots.AxesSubplot

plot_event_view(*return_per=(10, 25, 100)*, *axis=None*, ***kwargs*)

Plot averted damages for return periods. Call after calc().

Parameters

- **return_per** (*list, optional*) – years to visualize. Default 10, 25, 100
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5 (color is set by measures color attribute)

Return type

matplotlib.axes._subplots.AxesSubplot

static plot_waterfall(*hazard*, *entity*, *haz_future*, *ent_future*, *risk_func=<function risk_aai_agg>*, *axis=None*, ***kwargs*)

Plot waterfall graph at future with given risk metric. Can be called before and after calc().

Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.Entity*)
- **haz_future** (*Hazard*) – hazard in the future (future year provided at ent_future). `haz_future` is expected to have the same centroids as `hazard`.
- **ent_future** (*climada.Entity*) – entity in the future
- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

Return type

matplotlib.axes._subplots.AxesSubplot

plot_arrow_averted(*axis*, *in_meas_names=None*, *accumulate=False*, *combine=False*, *risk_func=<function risk_aai_agg>*, *disc_rates=None*, *imp_time_depen=1*, ***kwargs*)

Plot waterfall graph with accumulated values from present to future year. Call after calc() with save_imp=True.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot*) – axis from plot_waterfall or plot_waterfall_accumulated where arrow will be added to last bar
- **in_meas_names** (*list(str), optional*) – list with names of measures to represented total averted damage. Default: all measures
- **accumulate** (*bool, optional)*) – accumulated averted damage (True) or averted damage in future (False). Default: False
- **combine** (*bool, optional*) – use combine_measures to compute total averted damage (True) or just add benefits (False). Default: False
- **risk_func** (*func, optional*) – function describing risk measure given an Impact used in combine_measures. Default: average annual impact (aggregated).
- **disc_rates** (*DiscRates, optional*) – discount rates used in combine_measures
- **imp_time_depen** (*float, optional*) – parameter which represent time evolution of impact used in combine_measures. Default: 1 (linear).
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

plot_waterfall_accumulated(*hazard*, *entity*, *ent_future*, *risk_func=<function risk_aai_agg>*, *imp_time_depen=1*, *axis=None*, ***kwargs*)

Plot waterfall graph with accumulated values from present to future year. Call after calc() with save_imp=True. Provide same inputs as in calc.

Parameters

- **hazard** (*climada.Hazard*)
- **entity** (*climada.Entity*)
- **ent_future** (*climada.Entity*) – entity in the future
- **risk_func** (*func, optional*) – function describing risk measure given an Impact. Default: average annual impact (aggregated).
- **imp_time_depen** (*float, optional*) – parameter which represent time evolution of impact used in combine_measures. Default: 1 (linear).
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for bar matplotlib function, e.g. alpha=0.5

Return type
matplotlib.axes._subplots.AxesSubplot

climada.engine.cost_benefit.**risk_aai_agg**(*impact*)

Risk measurement as average annual impact aggregated.

Parameters
impact (*climada.engine.Impact*) – an Impact instance

Return type
float

climada.engine.cost_benefit.**risk_rp_100**(*impact*)

Risk measurement as exceedance impact at 100 years return period.

Parameters
impact (*climada.engine.Impact*) – an Impact instance

Return type
float

climada.engine.cost_benefit.**risk_rp_250**(*impact*)

Risk measurement as exceedance impact at 250 years return period.

Parameters
impact (*climada.engine.Impact*) – an Impact instance

Return type
float

climada.engine.forecast module

class climada.engine.forecast.**Forecast**(*hazard_dict: Dict[str,* Hazard*], exposure:* Exposures, *impact_funcs:* ImpactFuncSet, *haz_model: str = 'NWP'*, *exposure_name: str | None = None*)

Bases: object

Forecast definition. Compute an impact forecast with predefined hazard originating from a forecast (like numerical weather prediction models), exposure and impact. Use the calc() method to calculate a forecasted impact. Then use the plotting methods to illustrate the forecasted impacts. By default plots are saved under in a '/forecast/plots' folder in the configurable save_dir in local_data (see climada.util.config) under a name summarizing the Hazard type, haz model name, initialization time of the forecast run, event date, exposure name and the plot title. As the class is relatively new, there might be future changes to the attributes, the methods, and the parameters used to call the methods. It was discovered at some point, that there might be a memory leak in matplotlib even when figures are closed (https://github.com/matplotlib/matplotlib/issues/8519). Due to this reason the plotting functions in this module have the flag close_fig, to close figures within the function scope, which might mitigate that problem if a script runs this plotting functions many times.

run_datetime

initialization time of the forecast model run used to create the Hazard

Type
list of datetime.datetime

event_date

Date on which the Hazard event takes place

Type
datetime.datetime

hazard

List of the hazard forecast with different lead times.

Type
list of CLIMADA Hazard

haz_model

Short string specifying the model used to create the hazard, if possible three big letters.

Type
str

exposure

an CLIMADA Exposures containg values at risk

Type
Exposure

exposure_name

string specifying the exposure (e.g. 'EU'), which is used to name output files.

Type
str

vulnerability

Set of impact functions used in the impact calculation.

Type
ImpactFuncSet

__init__(*hazard_dict: Dict[str,* Hazard*]*, *exposure:* Exposures, *impact_funcs:* ImpactFuncSet, *haz_model: str = 'NWP'*, *exposure_name: str | None = None*)

Initialization with hazard, exposure and vulnerability.

Parameters

- **hazard_dict** (*dict*) – Dictionary of the format {run_datetime: Hazard} with run_datetime being the initialization time of a weather forecast run and Hazard being a CLIMADA Hazard derived from that forecast for one event. A probabilistic representation of that one event is possible, as long as the attribute Hazard.date is the same for all events. Several run_datetime:Hazard combinations for the same event can be provided.
- **exposure** (*Exposures*)
- **impact_funcs** (*ImpactFuncSet*)
- **haz_model** (*str, optional*) – Short string specifying the model used to create the hazard, if possible three big letters. Default is 'NWP' for numerical weather prediction.
- **exposure_name** (*str, optional*) – string specifying the exposure (e.g. 'EU'), which is used to name output files. If `None`, the name will be inferred from the Exposures GeoDataframe `region_id` column, using the corresponding name of the region with the lowest ISO 3166-1 numeric code. If that fails, it defaults to `"custom"`.

ei_exp(*run_datetime=None*)

Expected impact per exposure

Parameters
run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Return type
float

ai_agg(*run_datetime=None*)

average impact aggregated over all exposures

Parameters
run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Return type
float

haz_summary_str(*run_datetime=None*)

provide a summary string for the hazard part of the forecast

Parameters
run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Returns
summarizing the most important information about the hazard

Return type
str

summary_str(*run_datetime=None*)

provide a summary string for the impact forecast

Parameters
run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Returns
summarizing the most important information about the impact forecast

Return type
str

lead_time(*run_datetime=None*)

provide the lead time for the impact forecast

Parameters
run_datetime (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

Returns
the difference between the initialization time of the forecast model run and the date of the event, commenly named lead time

Return type
datetime.timedelta

calc(*force_reassign=False*)

calculate the impacts for all lead times using exposure, all hazards of all run_datetime, and ImpactFunctionSet.

Parameters
force_reassign (*bool, optional*) – Reassign hazard centroids to the exposure for all hazards, default is false.

plot_imp_map(*run_datetime=None*, *explain_str=None*, *save_fig=True*, *close_fig=False*, *polygon_file=None*, *polygon_file_crs='epsg:4326'*, *proj=<Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: Unknown based on WGS 84 ellipsoid - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *figsize=(9, 13)*, *adapt_fontsize=True*)

plot a map of the impacts

Parameters

- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.

- **explain_str** (*str, optional*) – Short str which explains type of impact, explain_str is included in the title of the figure. default is 'mean building damage caused by wind'
- **save_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save_dir and filename is derived from the method summary_str() (for more details see class docstring). Default is True.
- **close_fig** (*bool, optional*) – Figure not drawn if True. Default is False.
- **polygon_file** (*str, optional*) – Points to a .shp-file with polygons do be drawn as outlines on the plot, default is None to not draw the lines. please also specify the crs in the parameter polygon_file_crs.
- **polygon_file_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by pyproj.Proj. Default is 'epsg:4326'.
- **proj** (*ccrs*) – coordinate reference system used in coordinates The default is ccrs.PlateCarree()
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.

Returns
axes

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hist(*run_datetime=None*, *explain_str=None*, *save_fig=True*, *close_fig=False*, *figsize=(9, 8)*)

plot histogram of the forecasted impacts all ensemble members

Parameters

- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **explain_str** (*str, optional*) – Short str which explains type of impact, explain_str is included in the title of the figure. default is 'total building damage'
- **save_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save_dir and filename is derived from the method summary_str() (for more details see class docstring). Default is True.
- **close_fig** (*bool, optional*) – Figure is not drawn if True. Default is False.
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 8)

Returns
axes

Return type
matplotlib.axes.Axes

plot_exceedence_prob(*threshold*, *explain_str=None*, *run_datetime=None*, *save_fig=True*, *close_fig=False*, *polygon_file=None*, *polygon_file_crs='epsg:4326'*, *proj=<Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: Unknown based on WGS 84 ellipsoid - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *figsize=(9, 13)*, *adapt_fontsize=True*)

plot exceedence map

Parameters

- **threshold** (*float*) – Threshold of impact unit for which exceedence probability should be plotted.
- **explain_str** (*str, optional*) – Short str which explains threshold, explain_str is included in the title of the figure.
- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **save_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save_dir and filename is derived from the method summary_str() (for more details see class docstring). Default is True.
- **close_fig** (*bool, optional*) – Figure not drawn if True. Default is False.
- **polygon_file** (*str, optional*) – Points to a .shp-file with polygons do be drawn as outlines on the plot, default is None to not draw the lines. please also specify the crs in the parameter polygon_file_crs.
- **polygon_file_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by pyproj.Proj. Default is 'epsg:4326'.
- **proj** (*ccrs*) – coordinate reference system used in coordinates The default is ccrs.PlateCarree()
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise, the default matplotlib font size is used. Default is True.

Returns

axes

Return type

cartopy.mpl.geoaxes.GeoAxesSubplot

plot_warn_map(*polygon_file=None*, *polygon_file_crs='epsg:4326'*, *thresholds='default'*, *decision_level='exposure_point'*, *probability_aggregation=0.5*, *area_aggregation=0.5*, *title='WARNINGS'*, *explain_text='warn level based on thresholds'*, *run_datetime=None*, *proj=<Projected CRS: +proj=eqc +ellps=WGS84 +a=6378137.0 +lon_0=0.0 +to ...> Name: unknown Axis Info [cartesian]: - E[east]: Easting (unknown) - N[north]: Northing (unknown) - h[up]: Ellipsoidal height (metre) Area of Use: - undefined Coordinate Operation: - name: unknown - method: Equidistant Cylindrical Datum: Unknown based on WGS 84 ellipsoid - Ellipsoid: WGS 84 - Prime Meridian: Greenwich*, *figsize=(9, 13)*, *save_fig=True*, *close_fig=False*, *adapt_fontsize=True*)

plot map colored with 5 warning colors for all regions in provided shape file.

Parameters

- **polygon_file** (*str, optional*) – path to shp-file containing warning region polygons
- **polygon_file_crs** (*str, optional*) – String of pattern <provider>:<code> specifying the crs. has to be readable by pyproj.Proj. Default is 'epsg:4326'.
- **thresholds** (*list of 4 floats, optional*) – Thresholds for coloring region in second, third, forth and fifth warning color.

- **decision_level** (*str, optional*) – Either 'exposure_point' or 'polygon'. Default value is 'exposure_point'.
- **probability_aggregation** (*float or str, optional*) – Either a float between [0..1] spezifying a quantile or 'mean' or 'sum'. Default value is 0.5.
- **area_aggregation** (*float or str.*) – Either a float between [0..1] specifying a quantile or 'mean' or 'sum'. Default value is 0.5.
- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **title** (*str, optional*) – Default is 'WARNINGS'.
- **explain_text** (*str, optional*) – Defaut is 'warn level based on thresholds'.
- **proj** (*ccrs*) – coordinate reference system used in coordinates
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)
- **save_fig** (*bool, optional*) – Figure is saved if True, folder is within your configurable save_dir and filename is derived from the method summary_str() (for more details see class docstring). Default is True.
- **close_fig** (*bool, optional*) – Figure is not drawn if True. The default is False.
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise, the default matplotlib font size is used. Default is True.

Returns
axes

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin_ei_exposure(*run_datetime=None, figsize=(9, 13)*)

plot the expected impact

Parameters

- **run_datetime** (*datetime.datetime, optional*) – Select the used hazard by the run_datetime, default is first element of attribute run_datetime.
- **figsize** (*tuple*) – figure size for plt.subplots, width, height in inches The default is (9, 13)

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

climada.engine.impact module

class climada.engine.impact.**ImpactFreqCurve**(*return_per: ~numpy.ndarray = <factory>, impact: ~numpy.ndarray = <factory>, unit: str = '', frequency_unit: str = '1/year', label: str = ''*)

Bases: object

Impact exceedence frequency curve.

return_per: ndarray

return period

impact: ndarray

impact exceeding frequency

__init__(*return_per: ~numpy.ndarray = <factory>, impact: ~numpy.ndarray = <factory>, unit: str = '', frequency_unit: str = '1/year', label: str = ''*) → None

unit: str = ''

value unit used (given by exposures unit)

frequency_unit: str = '1/year'

value unit used (given by exposures unit)

label: str = ''

string describing source data

plot(*axis=None, log_frequency=False, **kwargs*)

Plot impact frequency curve.

Parameters

- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **log_frequency** (*boolean, optional*) – plot logarithmioc exceedance frequency on x-axis
- **kwargs** (*dict, optional*) – arguments for plot matplotlib function, e.g. color='b'

Return type

matplotlib.axes.Axes

class climada.engine.impact.**Impact**(*event_id=None, event_name=None, date=None, frequency=None, frequency_unit='1/year', coord_exp=None, crs='EPSG:4326', eai_exp=None, at_event=None, tot_value=0.0, aai_agg=0.0, unit='', imp_mat=None, haz_type=''*)

Bases: object

Impact definition. Compute from an entity (exposures and impact functions) and hazard.

event_id

id (>0) of each hazard event

Type

np.array

event_name

list name of each hazard event

Type

list

date

date if events as integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)

Type

np.array

coord_exp

exposures coordinates [lat, lon] (in degrees)

Type

np.array

crs

WKT string of the impact's crs

Type
str

eai_exp

expected impact for each exposure within a period of 1/frequency_unit

Type
np.array

at_event

impact for each hazard event

Type
np.array

frequency

frequency of event

Type
np.array

frequency_unit

frequency unit used (given by hazard), default is '1/year'

Type
str

aai_agg

average impact within a period of 1/frequency_unit (aggregated)

Type
float

unit

value unit used (given by exposures unit)

Type
str

imp_mat

matrix num_events x num_exp with impacts. only filled if save_mat is True in calc()

Type
sparse.csr_matrix

haz_type

the hazard type of the hazard

Type
str

__init__(*event_id=None*, *event_name=None*, *date=None*, *frequency=None*, *frequency_unit='1/year'*, *coord_exp=None*, *crs='EPSG:4326'*, *eai_exp=None*, *at_event=None*, *tot_value=0.0*, *aai_agg=0.0*, *unit=''*, *imp_mat=None*, *haz_type=''*)

Init Impact object

Parameters

- **event_id** (*np.array, optional*) – id (>0) of each hazard event

- **event_name** (*list, optional*) – list name of each hazard event
- **date** (*np.array, optional*) – date if events as integer date corresponding to the proleptic Gregorian ordinal, where January 1 of year 1 has ordinal 1 (ordinal format of datetime library)
- **frequency** (*np.array, optional*) – frequency of event impact for each hazard event
- **frequency_unit** (*np.array, optional*) – frequency unit, default: '1/year'
- **coord_exp** (*np.array, optional*) – exposures coordinates [lat, lon] (in degrees)
- **crs** (*Any, optional*) – Coordinate reference system. CRS instances from `pyproj` and `rasterio` will be transformed into WKT. Other types are not handled explicitly.
- **eai_exp** (*np.array, optional*) – expected impact for each exposure within a period of 1/frequency_unit
- **at_event** (*np.array, optional*) – impact for each hazard event
- **tot_value** (*float, optional*) – total exposure value affected
- **aai_agg** (*float, optional*) – average impact within a period of 1/frequency_unit (aggregated)
- **unit** (*str, optional*) – value unit used (given by exposures unit)
- **imp_mat** (*sparse.csr_matrix, optional*) – matrix num_events x num_exp with impacts.
- **haz_type** (*str, optional*) – the hazard type

calc(*exposures*, *impact_funcs*, *hazard*, *save_mat=False*, *assign_centroids=True*)

This function is deprecated, use `ImpactCalc.impact` instead.

classmethod from_eih(*exposures*, *hazard*, *at_event*, *eai_exp*, *aai_agg*, *imp_mat=None*)

Set Impact attributes from precalculated impact metrics.

Changed in version 3.3: The `impfset` argument was removed.

Parameters

- **exposures** (*climada.entity.Exposures*) – exposure used to compute imp_mat
- **impfset** (*climada.entity.ImpactFuncSet*) – impact functions set used to compute imp_mat
- **hazard** (*climada.Hazard*) – hazard used to compute imp_mat
- **at_event** (*np.array*) – impact for each hazard event
- **eai_exp** (*np.array*) – expected impact for each exposure within a period of 1/frequency_unit
- **aai_agg** (*float*) – average impact within a period of 1/frequency_unit (aggregated)
- **imp_mat** (*sparse.csr_matrix, optional*) – matrix num_events x num_exp with impacts. Default is None (empty sparse csr matrix).

Returns

impact with all risk metrics set based on the given impact matrix

Return type

climada.engine.impact.Impact

property tot_value

Return the total exposure value close to a hazard

Deprecated since version 3.3: Use `climada.entity.exposures.base.Exposures.affected_total_value()` instead.

transfer_risk(*attachment*, *cover*)

Compute the risk transfer for the full portfolio. This is the risk of the full portfolio summed over all events. For each event, the transfered risk amounts to the impact minus the attachment (but maximally equal to the cover) multiplied with the probability of the event.

Parameters

- **attachment** (*float*) – attachment per event for entire portfolio.
- **cover** (*float*) – cover per event for entire portfolio.

Returns

- **transfer_at_event** (*np.array*) – risk transfered per event
- **transfer_aai_agg** (*float*) – average risk within a period of 1/frequency_unit, transfered

residual_risk(*attachment*, *cover*)

Compute the residual risk after application of insurance attachment and cover to entire portfolio. This is the residual risk of the full portfolio summed over all events. For each event, the residual risk is obtained by subtracting the transfered risk from the trom the total risk per event. of the event.

Parameters

- **attachment** (*float*) – attachment per event for entire portfolio.
- **cover** (*float*) – cover per event for entire portfolio.

Returns

- **residual_at_event** (*np.array*) – residual risk per event
- **residual_aai_agg** (*float*) – average residual risk within a period of 1/frequency_unit

See also

transfer_risk
compute the transfer risk per portfolio.

calc_risk_transfer(*attachment*, *cover*)

Compute traaditional risk transfer over impact. Returns new impact with risk transfer applied and the insurance layer resulting Impact metrics.

Parameters

- **attachment** (*float*) – (deductible)
- **cover** (*float*)

Return type
climada.engine.impact.Impact

impact_per_year(*all_years=True*, *year_range=None*)

Calculate yearly impact from impact data.

Note: the impact in a given year is summed over all events. Thus, the impact in a given year can be larger than the total affected exposure value.

Parameters

- **all_years** (*boolean, optional*) – return values for all years between first and last year with event, including years without any events. Default: True

- **year_range** (*tuple or list with integers, optional*) – start and end year

Returns
year_set – Key=year, value=Summed impact per year.

Return type
dict

impact_at_reg(*agg_regions=None*)

Aggregate impact on given aggregation regions. This method works only if Impact.imp_mat was stored during the impact calculation.

Parameters
agg_regions (*np.array, list (optional)*) – The length of the array must equal the number of centroids in exposures. It reports what macro-regions these centroids belong to. For example, asuming there are three centroids and agg_regions = ['A', 'A', 'B'] then impact of the first and second centroids will be assigned to region A, whereas impact from the third centroid will be assigned to area B. If no aggregation regions are passed, the method aggregates impact at the country (admin_0) level. Default is None.

Returns
Contains the aggregated data per event. Rows: Hazard events. Columns: Aggregation regions.

Return type
pd.DataFrame

calc_impact_year_set(*all_years=True*, *year_range=None*)

This function is deprecated, use Impact.impact_per_year instead.

local_exceedance_impact(*return_periods=(25, 50, 100, 250)*, *method='interpolate'*, *min_impact=0*, *log_frequency=True*, *log_impact=True*, *bin_decimals=None*)

Compute local exceedance impact for given return periods. The default method is fitting the ordered impacts per centroid to the corresponding cummulated frequency with linear interpolation on log-log scale.

Parameters

- **return_periods** (*array_like*) – User-specified return periods for which the exceedance intensity should be calculated locally (at each centroid). Defaults to (25, 50, 100, 250).
- **method** (*str*) – Method to interpolate to new return periods. Currently available are "interpolate", "extrapolate", "extrapolate_constant" and "stepfunction". If set to "interpolate", return periods outside the range of the Impact object's observed local return periods will be assigned NaN. If set to "extrapolate_constant" or "stepfunction", return periods larger than the Impact object's observed local return periods will be assigned the largest local impact, and return periods smaller than the Impact object's observed local return periods will be assigned 0. If set to "extrapolate", local exceedance impacts will be extrapolated (and interpolated). The extrapolation to large return periods uses the two highest impacts of the centroid and their return periods and extends the interpolation between these points to the given return period (similar for small return periods). Defauls to "interpolate".
- **min_impact** (*float, optional*) – Minimum threshold to filter the impact. Defaults to 0.
- **log_frequency** (*bool, optional*) – If set to True, (cummulative) frequency values are converted to log scale before inter- and extrapolation. Defaults to True.
- **log_impact** (*bool, optional*) – If set to True, impact values are converted to log scale before inter- and extrapolation. Defaults to True.
- **bin_decimals** (*int, optional*) – Number of decimals to group and bin impact values. Binning results in smoother (and coarser) interpolation and more stable extrapolation. For more de-

tails and sensible values for bin_decimals, see Notes. If None, values are not binned. Defaults to None.

Returns

- **gdf** (*gpd.GeoDataFrame*) – GeoDataFrame containing exeedance impacts for given return periods. Each column corresponds to a return period, each row corresponds to a centroid. Values in the gdf correspond to the exceedance impact for the given centroid and return period
- **label** (*str*) – GeoDataFrame label, for reporting and plotting
- **column_label** (*function*) – Column-label-generating function, for reporting and plotting

See also

`util.interpolation.preprocess_and_interpolate_ev`
inter- and extrapolation method

Notes

If an integer bin_decimals is given, the impact values are binned according to their bin_decimals decimals, and their corresponding frequencies are summed. This binning leads to a smoother (and coarser) interpolation, and a more stable extrapolation. For instance, if bin_decimals=1, the two values 12.01 and 11.97 with corresponding frequencies 0.1 and 0.2 are combined to a value 12.0 with frequency 0.3. The default bin_decimals=None results in not binning the values. E.g., if your impact range from 1 to 100, you could use bin_decimals=1, if your impact range from 1e6 to 1e9, you could use bin_decimals=-5, if your impact range from 0.0001 to .01, you could use bin_decimals=5.

local_exceedance_imp(*return_periods=(25, 50, 100, 250)*)

This function is deprecated, use Impact.local_exceedance_impact instead.

Deprecated since version The: use of Impact.local_exceedance_imp is deprecated. Use Impact.local_exceedance_impact instead. Some errors in the previous calculation in Impact.local_exceedance_imp have been corrected. To reproduce data with the previous calculation, use CLIMADA v5.0.0 or less.

local_return_period(*threshold_impact=(1000.0, 10000.0)*, *method='interpolate'*, *min_impact=0*, *log_frequency=True*, *log_impact=True*, *bin_decimals=None*)

Compute local return periods for given threshold impacts. The default method is fitting the ordered impacts per centroid to the corresponding cummulated frequency with linear interpolation on log-log scale.

Parameters

- **threshold_impact** (*array_like*) – User-specified impact values for which the return period should be calculated locally (at each centroid). Defaults to (1000, 10000)
- **method** (*str*) – Method to interpolate to new threshold impacts. Currently available are "interpolate", "extrapolate", "extrapolate_constant" and "stepfunction". If set to "interpolate", threshold impacts outside the range of the Impact object's local impacts will be assigned NaN. If set to "extrapolate_constant" or "stepfunction", threshold impacts larger than the Impacts object's local impacts will be assigned NaN, and threshold impacts smaller than the Impact object's local impacts will be assigned the smallest observed local return period. If set to "extrapolate", local return periods will be extrapolated (and interpolated). The extrapolation to large threshold impacts uses the two highest impacts of the centroid and their return periods and extends the interpolation between these points to the given threshold imapct (similar for small large threshold impacts). Defaults to "interpolate".

- **min_impacts** (*float, optional*) – Minimum threshold to filter the impact. Defaults to 0.
- **log_frequency** (*bool, optional*) – If set to True, (cummulative) frequency values are converted to log scale before inter- and extrapolation. Defaults to True.
- **log_impact** (*bool, optional*) – If set to True, impact values are converted to log scale before inter- and extrapolation. Defaults to True.
- **bin_decimals** (*int, optional*) – Number of decimals to group and bin impact values. Binning results in smoother (and coarser) interpolation and more stable extrapolation. For more details and sensible values for bin_decimals, see Notes. If None, values are not binned. Defaults to None.

Returns

- **gdf** (*gpd.GeoDataFrame*) – GeoDataFrame containing return periods for given threshold impacts. Each column corresponds to a threshold_impact value, each row corresponds to a centroid. Values in the gdf correspond to the return period for the given centroid and threshold_impact value
- **label** (*str*) – GeoDataFrame label, for reporting and plotting
- **column_label** (*function*) – Column-label-generating function, for reporting and plotting

See also

`util.interpolation.preprocess_and_interpolate_ev`
inter- and extrapolation method

Notes

If an integer bin_decimals is given, the impact values are binned according to their bin_decimals decimals, and their corresponding frequencies are summed. This binning leads to a smoother (and coarser) interpolation, and a more stable extrapolation. For instance, if bin_decimals=1, the two values 12.01 and 11.97 with corresponding frequencies 0.1 and 0.2 are combined to a value 12.0 with frequency 0.3. The default bin_decimals=None results in not binning the values. E.g., if your impact range from 1 to 100, you could use bin_decimals=1, if your impact range from 1e6 to 1e9, you could use bin_decimals=-5, if your impact range from 0.0001 to .01, you could use bin_decimals=5.

calc_freq_curve(*return_per=None*)

Compute impact exceedance frequency curve.

Parameters

return_per (*np.array, optional*) – return periods where to compute the exceedance impact. Use impact's frequencies if not provided

Return type

ImpactFreqCurve

plot_scatter_eai_exposure(*mask=None, ignore_zero=False, pop_name=True, buffer=0.0, extend='neither', axis=None, adapt_fontsize=True, **kwargs*)

Plot scatter expected impact within a period of 1/frequency_unit of each exposure.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False

- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 1.0.
- **extend** (*str*) – optional extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*dict, optional*) – arguments for hexbin matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin_eai_exposure(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *axis=None*, *adapt_fontsize=True*, ***kwargs*)

Plot hexbin expected impact within a period of 1/frequency_unit of each exposure.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 1.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default: True
- **kwargs** (*dict, optional*) – arguments for hexbin matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_raster_eai_exposure(*res=None*, *raster_res=None*, *save_tiff=None*, *raster_f=<function Impact.<lambda>>*, *label='value (log10)'*, *axis=None*, *adapt_fontsize=True*, ***kwargs*)

Plot raster expected impact within a period of 1/frequency_unit of each exposure.

Parameters

- **res** (*float, optional*) – resolution of current data in units of latitude and longitude, approximated if not provided.
- **raster_res** (*float, optional*) – desired resolution of the raster
- **save_tiff** (*str, optional*) – file name to save the raster in tiff format, if provided
- **raster_f** (*lambda function*) – transformation to use to data. Default: log10 adding 1.
- **label** (*str*) – colorbar label
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*dict, optional*) – arguments for imshow matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_basemap_eai_exposure(*mask=None, ignore_zero=False, pop_name=True, buffer=0.0, extend='neither', zoom=10, url={'attribution': '(C) OpenStreetMap contributors (C) CARTO', 'html_attribution': '© OpenStreetMap contributors © CARTO', 'max_zoom': 20, 'name': 'CartoDB.Positron', 'subdomains': 'abcd', 'url': 'https://{s}.basemaps.cartocdn.com/{variant}/{z}/{x}/{y}{r}.png', 'variant': 'light_all'}, axis=None, **kwargs*)

Plot basemap expected impact of each exposure within a period of 1/frequency_unit.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*str, optional*) – image source, default: ctx.providers.CartoDB.Positron
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **kwargs** (*dict, optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'. Default: 'Wistia'

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin_impact_exposure(*event_id=1, mask=None, ignore_zero=False, pop_name=True, buffer=0.0, extend='neither', axis=None, adapt_fontsize=True, **kwargs*)

Plot hexbin impact of an event at each exposure. Requires attribute imp_mat.

Parameters

- **event_id** (*int, optional*) – id of the event for which to plot the impact. Default: 1.
- **mask** (*np.array, optional*) – mask to apply to impact plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 1.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **axis** (*matplotlib.axes.Axes*) – optional axis to use
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*dict, optional*) – arguments for hexbin matplotlib function

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_basemap_impact_exposure(*event_id=1, mask=None, ignore_zero=False, pop_name=True, buffer=0.0, extend='neither', zoom=10, url={'attribution': '(C) OpenStreetMap contributors (C) CARTO', 'html_attribution': '© OpenStreetMap contributors © CARTO', 'max_zoom': 20, 'name': 'CartoDB.Positron', 'subdomains': 'abcd', 'url': 'https://{s}.basemaps.cartocdn.com/{variant}/{z}/{x}/{y}{r}.png', 'variant': 'light_all'}, axis=None, **kwargs*)

Plot basemap impact of an event at each exposure. Requires attribute imp_mat.

Parameters

- **event_id** (*int, optional*) – id of the event for which to plot the impact. Default: 1.
- **mask** (*np.array, optional*) – mask to apply to impact plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*str, optional*) – image source, default: ctx.providers.CartoDB.Positron
- **axis** (*matplotlib.axes.Axes, optional*) – axis to use
- **kwargs** (*dict, optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'. Default: 'Wistia'

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_rp_imp(*return_periods=(25, 50, 100, 250), log10_scale=True, axis=None, mask_distance=0.03, kwargs_local_exceedance_impact=None, **kwargs*)

Compute and plot exceedance impact maps for different return periods. Calls local_exceedance_impact. For handling large data sets and for further options, see Notes.

Parameters

- **return_periods** (*tuple of int, optional*) – return periods to consider. Default: (25, 50, 100, 250)
- **log10_scale** (*boolean, optional*) – plot impact as log10(impact). Default: True
- **smooth** (*bool, optional*) – smooth plot to plot.RESOLUTIONxplot.RESOLUTION. Default: True
- **mask_distance** (*float, optional*) – Only regions are plotted that are closer to any of the data points than this distance, relative to overall plot size. For instance, to only plot values at the centroids, use mask_distance=0.03. If None, the plot is not masked. Default is 0.03.
- **kwargs_local_exceedance_impact** (*dict*) – Dictionary of keyword arguments for the method impact.local_exceedance_impact.
- **kwargs** (*dict, optional*) – arguments for pcolormesh matplotlib function used in event plots

Returns

- **axis** (*matplotlib.axes.Axes*)
- **imp_stats** (*np.array*) – return_periods.size x num_centroids

See also

engine.impact.local_exceedance_impact
inter- and extrapolation method

Notes

For handling large data, and for more flexible options in the exceedance impact computation and in the plotting, we recommend to use gdf, title, labels = impact.local_exceedance_impact() and util.plot.plot_from_gdf(gdf, title, labels) instead.

write_csv (*file_name*)

Write data into csv file. imp_mat is not saved.

Parameters
file_name (*str*) – absolute path of the file

write_excel (*file_name*)

Write data into Excel file. imp_mat is not saved.

Parameters
file_name (*str*) – absolute path of the file

write_hdf5 (*file_path: str | Path, dense_imp_mat: bool = False*)

Write the data stored in this object into an H5 file.

Try to write all attributes of this class into H5 datasets or attributes. By default, any iterable will be stored in a dataset and any string or scalar will be stored in an attribute. Dictionaries will be stored as groups, with the previous rules being applied recursively to their values.

The impact matrix can be stored in a sparse or dense format.

Parameters

- **file_path** (*str or Path*) – File path to write data into. The enclosing directory must exist.
- **dense_imp_mat** (*bool*) – If `True`, write the impact matrix as dense matrix that can be more easily interpreted by common H5 file readers but takes up (vastly) more space. Defaults to `False`.

Raises
TypeError – If `event_name` does not contain strings exclusively.

write_sparse_csr (*file_name*)

Write imp_mat matrix in numpy's npz format.

static read_sparse_csr (*file_name*)

Read imp_mat matrix from numpy's npz format.

Parameters
file_name (*str*)

Return type
sparse.csr_matrix

classmethod from_csv(*file_name*)

Read csv file containing impact data generated by write_csv.

Parameters
file_name (*str*) – absolute path of the file

Returns
imp – Impact from csv file

Return type
climada.engine.impact.Impact

read_csv(**args*, ***kwargs*)

This function is deprecated, use Impact.from_csv instead.

classmethod from_excel(*file_name*)

Read excel file containing impact data generated by write_excel.

Parameters
file_name (*str*) – absolute path of the file

Returns
imp – Impact from excel file

Return type
climada.engine.impact.Impact

read_excel(**args*, ***kwargs*)

This function is deprecated, use Impact.from_excel instead.

classmethod from_hdf5(*file_path: str | Path*)

Create an impact object from an H5 file.

This assumes a specific layout of the file. If values are not found in the expected places, they will be set to the default values for an `Impact` object.

The following H5 file structure is assumed (H5 groups are terminated with `/`, attributes are denoted by `.attrs/`):

```
file.h5
├─ at_event
├─ coord_exp
├─ eai_exp
├─ event_id
├─ event_name
├─ frequency
├─ imp_mat
├─ .attrs/
│  ├─ aai_agg
│  ├─ crs
│  ├─ frequency_unit
│  ├─ haz_type
│  ├─ tot_value
│  ├─ unit
```

As per the `climada.engine.impact.Impact.__init__()`, any of these entries is optional. If it is not found, the default value will be used when constructing the Impact.

The impact matrix `imp_mat` can either be an H5 dataset, in which case it is interpreted as dense representation of the matrix, or an H5 group, in which case the group is expected to contain the following data for instantiating a scipy.sparse.csr_matrix:

```
imp_mat/
├─ data
├─ indices
├─ indptr
├─ .attrs/
│  ├─ shape
```

Parameters
file_path (*str or Path*) – The file path of the file to read.

Returns
imp – Impact with data from the given file

Return type
Impact

static video_direct_impact(*exp*, *impf_set*, *haz_list*, *file_name=''*, *writer=<matplotlib.animation.PillowWriter object>*, *imp_thresh=0*, *args_exp=None*, *args_imp=None*, *ignore_zero=False*, *pop_name=False*)

Computes and generates video of accumulated impact per input events over exposure.

Parameters

- **exp** (*climada.entity.Exposures*) – exposures instance, constant during all video
- **impf_set** (*climada.entity.ImpactFuncSet*) – impact functions
- **haz_list** ((*list(Hazard)*)) – every Hazard contains an event; all hazards use the same centroids
- **file_name** (*str, optional*) – file name to save video, if provided
- **writer** (*matplotlib.animation.*, optional*) – video writer. Default: pillow with bitrate=500
- **imp_thresh** (*float, optional*) – represent damages greater than threshold. Default: 0
- **args_exp** (*dict, optional*) – arguments for scatter (points) or hexbin (raster) matplotlib function used in exposures
- **args_imp** (*dict, optional*) – arguments for scatter (points) or hexbin (raster) matplotlib function used in impact
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places The default is False.

Return type
list of *Impact*

select(*event_ids=None*, *event_names=None*, *dates=None*, *coord_exp=None*, *reset_frequency=False*)

Select a subset of events and/or exposure points from the impact. If multiple input variables are not None, it returns all the impacts matching at least one of the conditions.

Notes

the frequencies are NOT adjusted. Method to adjust frequencies

and obtain correct eai_exp:

1- Select subset of impact according to your choice imp = impact.select(…) 2- Adjust manually the frequency of the subset of impact imp.frequency = […] 3- Use select without arguments to select all events and recompute the eai_exp with the updated frequencies. imp = imp.select()

Parameters

- **event_ids** (*list of int, optional*) – Selection of events by their id. The default is None.
- **event_names** (*list of str, optional*) – Selection of events by their name. The default is None.
- **dates** (*tuple, optional*) – (start-date, end-date), events are selected if they are >= than start-date and <= than end-date. Dates in same format as impact.date (ordinal format of datetime library) The default is None.
- **coord_exp** (*np.array, optional*) – Selection of exposures coordinates [lat, lon] (in degrees) The default is None.
- **reset_frequency** (*bool, optional*) – Change frequency of events proportional to difference between first and last year (old and new). Assumes annual frequency values. Default: False.

Raises

`ValueError` – If the impact matrix is missing, the eai_exp and aai_agg cannot be updated for a selection of events and/or exposures.

Returns

imp – A new impact object with a selection of events and/or exposures

Return type

climada.engine.impact.Impact

`classmethod concat`(*imp_list: Iterable*, *reset_event_ids: bool = False*)

Concatenate impact objects with the same exposure

This function is useful if, e.g. different impact functions have to be applied for different seasons (e.g. for agricultural impacts).

It checks if the exposures of the passed impact objects are identical and then

- concatenates the attributes `event_id`, `event_name`, `date`, `frequency`, `imp_mat`, `at_event`,
- sums up the values of attributes `eai_exp`, `aai_exp`
- and takes the following attributes from the first impact object in the passed impact list: `coord_exp`, `crs`, `unit`, `tot_value`, `frequency_unit`, `haz_type`

If event ids are not unique among the passed impact objects an error is raised. In this case, the user can set `reset_event_ids=True` to create unique event ids for the concatenated impact.

If all impact matrices of the impacts in `imp_list` are empty, the impact matrix of the concatenated impact is also empty.

Parameters

- **imp_list** (*Iterable of climada.engine.impact.Impact*) – Iterable of Impact objects to concatenate
- **reset_event_ids** (*boolean, optional*) – Reset event ids of the concatenated impact object

Returns
impact – New impact object which is a concatenation of all impacts

Return type
climada.engine.impact.Impact

Notes

- Concatenation of impacts with different exposure (e.g. different countries) could also be implemented here in the future.

match_centroids(*hazard*, *distance='euclidean'*, *threshold=100*)

Finds the closest hazard centroid for each impact coordinate. Creates a temporary GeoDataFrame and uses `u_coord.match_centroids()`. See there for details and parameters

Parameters

- **hazard** (*Hazard*) – Hazard to match (with raster or vector centroids).
- **distance** (*str, optional*) – Distance to use in case of vector centroids. Possible values are "euclidean", "haversine" and "approx". Default: "euclidean"
- **threshold** (*float*) – If the distance (in km) to the nearest neighbor exceeds *threshold*, the index *-1* is assigned. Set *threshold* to 0, to disable nearest neighbor matching. Default: 100 (km)

Returns
array of closest Hazard centroids, aligned with the Impact's *coord_exp* array

Return type
np.array

climada.engine.impact_calc module

class climada.engine.impact_calc.**ImpactCalc**(*exposures*, *impfset*, *hazard*)

Bases: `object`

Class to compute impacts from exposures, impact function set and hazard

__init__(*exposures*, *impfset*, *hazard*)

ImpactCalc constructor

The dimension of the imp_mat variable must be compatible with the exposures and hazard objects.

This will call *climada.hazard.base.Hazard.check_matrices()*.

Parameters

- **exposures** (*climada.entity.Exposures*) – exposures used to compute impacts
- **impf_set** (*climada.entity.ImpactFuncSet*) – impact functions set used to compute impacts
- **hazard** (*climada.Hazard*) – hazard used to compute impacts

property n_exp_pnt

Number of exposure points (rows in gdf)

property n_events

Number of hazard events (size of event_id array)

impact (*save_mat=True*, *assign_centroids=True*, *ignore_cover=False*, *ignore_deductible=False*)

Compute the impact of a hazard on exposures.

Parameters

- **save_mat** (*bool, optional*) – if true, save the total impact matrix (events x exposures) Default: True
- **assign_centroids** (*bool, optional*) – indicates whether centroids are assigned to the self.exposures object. The default centroids assignment uses Euclidean distance and a threshold equal to twice the highest resolution (smallest distance between any two nearest neighbours) of the centroids (assuming a regular grid). Recommendation: assign the centroids directly with control over the distance metric and the distance threshold using *exposure.assign_centroids(hazard)*. In this case, set assign_centroids to `False`. Centroids assignment is an expensive operation; set this to `False` to save computation time if the hazards' centroids are already assigned to the exposures object. Default: True
- **ignore_cover** (*bool, optional*) – if set to True, the column 'cover' of the exposures GeoDataFrame, if present, is ignored and the impact it not capped by the values in this column. Default: False
- **ignore_deductible** (*bool, opotional*) – if set to True, the column 'deductible' of the exposures GeoDataFrame, if present, is ignored and the impact it not reduced through values in this column. Default: False

Examples

```
>>> haz = Hazard.from_hdf5(HAZ_DEMO_H5)  # Set hazard
>>> impfset = ImpactFuncSet.from_excel(ENT_TEMPLATE_XLS)
>>> exp = Exposures(pd.read_excel(ENT_TEMPLATE_XLS))
>>> impcalc = ImpactCal(exp, impfset, haz)
>>> imp = impcalc.impact(insured=True)
>>> imp.aai_agg
```

```
>>> haz = Hazard.from_hdf5(HAZ_DEMO_H5)  # Set hazard
>>> impfset = ImpactFuncSet.from_excel(ENT_TEMPLATE_XLS)
>>> exp = Exposures(pd.read_excel(ENT_TEMPLATE_XLS))
>>> #Adjust threshold and distance to centroids/exposure resolution
>>> exp.assign_centroids(hazard, threshold=1.5, distance='euclidean')
>>> impcalc = ImpactCal(exp, impfset, haz)
>>> imp = impcalc.impact(insured=True, assign_centroids=False)
>>> imp.aai_agg
```

See also

apply_deductible_to_mat
apply deductible to impact matrix

apply_cover_to_mat
apply cover to impact matrix

climada.entity.exposures.assign_centroids
assign centroids to exposures explicitly

minimal_exp_gdf(*impf_col*, *assign_centroids*, *ignore_cover*, *ignore_deductible*)

Get minimal exposures geodataframe for impact computation

Parameters

- **exposures** (*climada.entity.Exposures*)
- **hazard** (*climada.Hazard*)
- **impf_col** (*str*) – Name of the impact function column in exposures.gdf
- **assign_centroids** (*bool*) – Indicates whether centroids are re-assigned to the self.exposures object or kept from previous impact calculation with a hazard of the same hazard type. Centroids assignment is an expensive operation; set this to `False` to save computation time if the centroids have not changed since the last impact calculation.
- **include_cover** (*bool*) – if set to True, the column 'cover' of the exposures GeoDataFrame is excluded from the returned GeoDataFrame, otherwise it is included if present.
- **include_deductible** (*bool*) – if set to True, the column 'deductible' of the exposures GeoDataFrame is excluded from the returned GeoDataFrame, otherwise it is included if present.

imp_mat_gen(*exp_gdf*, *impf_col*)

Generator of impact sub-matrices and correspoding exposures indices

The exposures gdf is decomposed into chunks that fit into the max defined memory size. For each chunk, the impact matrix is computed and returned, together with the corresponding exposures points index.

Parameters

- **exp_gdf** (*GeoDataFrame*) – Geodataframe of the exposures with columns required for impact computation.
- **impf_col** (*str*) – name of the desired impact column in the exposures.

Raises

ValueError – if the hazard is larger than the memory limit

Yields

scipy.sparse.crs_matrix, np.ndarray – impact matrix and corresponding exposures indices for each chunk.

insured_mat_gen(*imp_mat_gen*, *exp_gdf*, *impf_col*)

Generator of insured impact sub-matrices (with applied cover and deductible) and corresponding exposures indices

This generator takes a 'regular' impact matrix generator and applies cover and deductible onto the impacts. It yields the same sub-matrices as the original generator.

Deductible and cover are taken from the dataframe stored in *exposures.gdf*.

Parameters

- **imp_mat_gen** (*generator of tuples (sparse.csr_matrix, np.array)*) – The generator for creating the impact matrix. It returns a part of the full matrix and the associated exposure indices.
- **exp_gdf** (*GeoDataFrame*) – Geodataframe of the exposures with columns required for impact computation.
- **impf_col** (*str*) – Name of the column in 'exp_gdf' indicating the impact function (id)

Yields

- **mat** (*scipy.sparse.csr_matrix*) – Impact sub-matrix (with applied cover and deductible) with size (n_events, len(exp_idx))

- **exp_idx** (*np.array*) – Exposure indices for impacts in mat

impact_matrix(*exp_values*, *cent_idx*, *impf*)

Compute the impact matrix for given exposure values, assigned centroids, a hazard, and one impact function.

Parameters

- **exp_values** (*np.array*) – Exposure values
- **cent_idx** (*np.array*) – Hazard centroids assigned to each exposure location
- **hazard** (*climada.Hazard*) – Hazard object
- **impf** (*climada.entity.ImpactFunc*) – one impactfunction comon to all exposure elements in exp_gdf

Returns
Impact per event (rows) per exposure point (columns)

Return type
scipy.sparse.csr_matrix

stitch_impact_matrix(*imp_mat_gen*)

Make an impact matrix from an impact sub-matrix generator

stitch_risk_metrics(*imp_mat_gen*)

Compute the impact metrics from an impact sub-matrix generator

This method is used to compute the risk metrics if the user decided not to store the full impact matrix.

Parameters
imp_mat_gen (*generator of tuples (sparse.csr_matrix, np.array)*) – The generator for creating the impact matrix. It returns a part of the full matrix and the associated exposure indices.

Returns

- **at_event** (*np.array*) – Accumulated damage for each event
- **eai_exp** (*np.array*) – Expected impact within a period of 1/frequency_unit for each exposure point
- **aai_agg** (*float*) – Average impact within a period of 1/frequency_unit aggregated

static apply_deductible_to_mat(*mat*, *deductible*, *hazard*, *cent_idx*, *impf*)

Apply a deductible per exposure point to an impact matrix at given centroid points for given impact function.

All exposure points must have the same impact function. For different impact functions apply use this method repeatedly on the same impact matrix.

Parameters

- **imp_mat** (*scipy.sparse.csr_matrix*) – impact matrix (events x exposure points)
- **deductible** (*np.array()*) – deductible for each exposure point
- **hazard** (*climada.Hazard*) – hazard used to compute the imp_mat
- **cent_idx** (*np.array()*) – index of centroids associated with each exposure point
- **impf** (*climada.entity.ImpactFunc*) – impact function associated with the exposure points

Returns
imp_mat – impact matrix with applied deductible

Return type
scipy.sparse.csr_matrix

static apply_cover_to_mat (*mat*, *cover*)

Apply cover to impact matrix.

The impact data is clipped to the range [0, cover]. The cover is defined per exposure point.

Parameters

- **imp_mat** (*scipy.sparse.csr_matrix*) – impact matrix
- **cover** (*np.array()*) – cover per exposures point (columns of imp_mat)

Returns
imp_mat – impact matrix with applied cover

Return type
scipy.sparse.csr_matrix

static eai_exp_from_mat (*mat*, *freq*)

Compute impact for each exposures from the total impact matrix

Parameters

- **imp_mat** (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.
- **frequency** (*np.array*) – frequency of events within a period of 1/frequency_unit

Returns
eai_exp – expected impact within a period of 1/frequency_unit for each exposure

Return type
np.array

static at_event_from_mat (*mat*)

Compute impact for each hazard event from the total impact matrix :Parameters: **imp_mat** (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.

Returns
at_event – impact for each hazard event

Return type
np.array

static aai_agg_from_eai_exp (*eai_exp*)

Aggregate impact.eai_exp

Parameters
eai_exp (*np.array*) – expected impact within a period of 1/frequency_unit for each exposure point

Returns
average aggregated impact within a period of 1/frequency_unit

Return type
float

classmethod risk_metrics (*mat*, *freq*)

Compute risk metricss eai_exp, at_event, aai_agg for an impact matrix and a frequency vector.

Parameters

- **mat** (*sparse.csr_matrix*) – matrix num_events x num_exp with impacts.
- **freq** (*np.array*) – array with the frequency per event

Returns

- **eai_exp** (*np.array*) – expected impact within a period of 1/frequency_unit at each exposure point
- **at_event** (*np.array()*) – total impact for each event
- **aai_agg** (*float*) – average impact within a period of 1/frequency_unit aggregated over all exposure points

climada.engine.impact_data module

climada.engine.impact_data.**assign_hazard_to_emdat**(*certainty_level*, *intensity_path_haz*, *names_path_haz*, *reg_id_path_haz*, *date_path_haz*, *emdat_data*, *start_time*, *end_time*, *keep_checks=False*)

assign_hazard_to_emdat: link EMdat event to hazard

Parameters

- **certainty_level** (*str*) – 'high' or 'low'
- **intensity_path_haz** (*sparse matrix*) – with hazards as rows and grid points as cols, values only at location with impacts
- **names_path_haz** (*str*) – identifier for each hazard (i.e. IBtracID) (rows of the matrix)
- **reg_id_path_haz** (*str*) – ISO country ID of each grid point (cols of the matrix)
- **date_path_haz** (*str*) – start date of each hazard (rows of the matrix)
- **emdat_data** (*pd.DataFrame*) – dataframe with EMdat data
- **start_time** (*str*) – start date of events to be assigned 'yyyy-mm-dd'
- **end_time** (*str*) – end date of events to be assigned 'yyyy-mm-dd'
- **keep_checks** (*bool, optional*)

Return type
pd.dataframe with EMdat entries linked to a hazard

climada.engine.impact_data.**hit_country_per_hazard**(*intensity_path*, *names_path*, *reg_id_path*, *date_path*)

hit_country_per_hazard: create list of hit countries from hazard set

Parameters

- **intensity_path** (*str*) – Path to file containing sparse matrix with hazards as rows and grid points as cols, values only at location with impacts
- **names_path** (*str*) – Path to file with identifier for each hazard (i.e. IBtracID) (rows of the matrix)
- **reg_id_path** (*str*) – Path to file with ISO country ID of each grid point (cols of the matrix)
- **date_path** (*str*) – Path to file with start date of each hazard (rows of the matrix)

Return type
pd.DataFrame with all hit countries per hazard

climada.engine.impact_data.**create_lookup**(*emdat_data*, *start*, *end*, *disaster_subtype='Tropical cyclone'*)

create_lookup: prepare a lookup table of EMdat events to which hazards can be assigned

Parameters

- **emdat_data** (*pd.DataFrame*) – with EMdat data

- **start** (*str*) – start date of events to be assigned 'yyyy-mm-dd'
- **end** (*str*) – end date of events to be assigned 'yyyy-mm-dd'
- **disaster_subtype** (*str*) – EMdat disaster subtype

Return type
pd.DataFrame

climada.engine.impact_data.**emdat_possible_hit**(*lookup*, *hit_countries*, *delta_t*)

relate EM disaster to hazard using hit countries and time

Parameters

- **lookup** (*pd.DataFrame*) – to relate EMdatID to hazard
- **delta_t** – max time difference of start of EMdat event and hazard
- **hit_countries**

Return type
list with possible hits

climada.engine.impact_data.**match_em_id**(*lookup*, *poss_hit*)

function to check if EM_ID has been assigned already and combine possible hits

Parameters

- **lookup** (*pd.dataframe*) – to relate EMdatID to hazard
- **poss_hit** (*list*) – with possible hits

Returns
with all possible hits per EMdat ID

Return type
list

climada.engine.impact_data.**assign_track_to_em**(*lookup*, *possible_tracks_1*, *possible_tracks_2*, *level*)

function to assign a hazard to an EMdat event to get some confidene into the procedure, hazards get only assigned if there is no other hazard occuring at a bigger time interval in that country Thus a track of possible_tracks_1 gets only assigned if there are no other tracks in possible_tracks_2. The confidence can be expressed with a certainty level

Parameters

- **lookup** (*pd.DataFrame*) – to relate EMdatID to hazard
- **possible_tracks_1** (*list*) – list of possible hits with smaller time horizon
- **possible_tracks_2** (*list*) – list of possible hits with larger time horizon
- **level** (*int*) – level of confidence

Returns
lookup with assigend tracks and possible hits

Return type
pd.DataFrame

climada.engine.impact_data.**check_assigned_track**(*lookup*, *checkset*)

compare lookup with assigned tracks to a set with checked sets

Parameters

- **lookup** (*pd.DataFrame*) – dataframe to relate EMdatID to hazard

- **checkset** (*pd.DataFrame*) – dataframe with already checked hazards

Return type
error scores

climada.engine.impact_data.**clean_emdat_df**(*emdat_file*, *countries=None*, *hazard=None*, *year_range=None*, *target_version=None*)

Get a clean and standardized DataFrame from EM-DAT-CSV-file (1) load EM-DAT data from CSV to DataFrame and remove header/footer, (2) handle version, clean up, and add columns, and (3) filter by country, hazard type and year range (if any given)

Parameters

- **emdat_file** (*str, Path, or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV
- **countries** (*list of str*) – country ISO3-codes or names, e.g. ['JAM', 'CUB']. countries=None for all countries (default)
- **hazard** (*list or str*) – List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.
- **year_range** (*list or tuple*) – Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)
- **target_version** (*int*) – required EM-DAT data format version (i.e. year of download), changes naming of columns/variables, default: newest available version in `VARNAMES_EMDAT` that matches the given emdat_file

Returns
df_data – DataFrame containing cleaned and filtered EM-DAT impact data

Return type
pd.DataFrame

climada.engine.impact_data.**emdat_countries_by_hazard**(*emdat_file_csv*, *hazard=None*, *year_range=None*)

return list of all countries exposed to a chosen hazard type from EMDAT data as CSV.

Parameters

- **emdat_file** (*str, Path, or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV
- **hazard** (*list or str*) – List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.
- **year_range** (*list or tuple*) – Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

Returns

- **countries_iso3a** (*list*) – List of ISO3-codes of countries impacted by the disaster (sub-)types
- **countries_names** (*list*) – List of names of countries impacted by the disaster (sub-)types

climada.engine.impact_data.**scale_impact2refyear**(*impact_values*, *year_values*, *iso3a_values*, *reference_year=None*)

Scale give impact values proportional to GDP to the according value in a reference year (for normalization of monetary values)

Parameters

- **impact_values** (*list or array*) – Impact values to be scaled.
- **year_values** (*list or array*) – Year of each impact (same length as impact_values)
- **iso3a_values** (*list or array*) – ISO3alpha code of country for each impact (same length as impact_values)
- **reference_year** (*int, optional*) – Impact is scaled proportional to GDP to the value of the reference year. No scaling for reference_year=None (default)

climada.engine.impact_data.**emdat_impact_yearlysum**(*emdat_file_csv*, *countries=None*, *hazard=None*, *year_range=None*, *reference_year=None*, *imp_str="Total Damages ('000 US$)"*, *version=None*)

function to load EM-DAT data and sum impact per year

Parameters

emdat_file_csv (*str or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV

countries

[list of str] country ISO3-codes or names, e.g. ['JAM', 'CUB']. countries=None for all countries (default)

hazard

[list or str] List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

year_range

[list or tuple] Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

version

[int, optional] required EM-DAT data format version (i.e. year of download), changes naming of columns/variables, default: newest available version in `VARNAMES_EMDAT`

Returns

out – DataFrame with summed impact and scaled impact per year and country.

Return type

pd.DataFrame

climada.engine.impact_data.**emdat_impact_event**(*emdat_file_csv*, *countries=None*, *hazard=None*, *year_range=None*, *reference_year=None*, *imp_str="Total Damages ('000 US$)"*, *version=None*)

function to load EM-DAT data return impact per event

Parameters

emdat_file_csv (*str or DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV

countries
[list of str] country ISO3-codes or names, e.g. ['JAM', 'CUB']. default: countries=None for all countries

hazard
[list or str] List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.; OR CLIMADA hazard type abbreviations, e.g. TC, BF, etc.

year_range
[list or tuple] Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)

reference_year
[int reference year of exposures. Impact is scaled] proportional to GDP to the value of the reference year. Default: No scaling for 0

imp_str
[str] Column name of impact metric in EMDAT CSV, default = "Total Damages ('000 US$)"

version
[int, optional] EM-DAT version to take variable/column names from, default: newest available version in `VARNAMES_EMDAT`

Returns
out – EMDAT DataFrame with new columns "year", "region_id", and "impact" and +impact_scaled" total impact per event with same unit as chosen impact, but multiplied by 1000 if impact is given as 1000 US$ (e.g. imp_str="Total Damages ('000 US$) scaled").

Return type
pd.DataFrame

`climada.engine.impact_data.`**`emdat_to_impact`**(*emdat_file_csv*, *hazard_type_climada*, *year_range=None*, *countries=None*, *hazard_type_emdat=None*, *reference_year=None*, *imp_str='Total Damages'*)

function to load EM-DAT data return impact per event

Parameters

- **emdat_file_csv** (*str or pd.DataFrame*) – Either string with full path to CSV-file or pandas.DataFrame loaded from EM-DAT CSV
- **countries** (*list of str*) – country ISO3-codes or names, e.g. ['JAM', 'CUB']. default: countries=None for all countries
- **hazard_type_climada** (*str*) – CLIMADA hazard type abbreviations, e.g. TC, BF, etc.
- **hazard_type_emdat** (*list or str*) – List of Disaster (sub-)type accordung EMDAT terminology, i.e.: Animal accident, Drought, Earthquake, Epidemic, Extreme temperature, Flood, Fog, Impact, Insect infestation, Landslide, Mass movement (dry), Storm, Volcanic activity, Wildfire; Coastal Flooding, Convective Storm, Riverine Flood, Tropical cyclone, Tsunami, etc.
- **year_range** (*list or tuple*) – Year range to be extracted, e.g. (2000, 2015); (only min and max are considered)
- **reference_year** (*int reference year of exposures. Impact is scaled*) – proportional to GDP to the value of the reference year. Default: No scaling for 0
- **imp_str** (*str*) – Column name of impact metric in EMDAT CSV, default = "Total Damages ('000 US$)"

Returns

- **impact_instance** (*climada.engine.Impact*) – impact object of same format as output from CLIMADA impact computation. Values scaled with GDP to reference_year if reference_year is given. i.e. current US$ for imp_str="Total Damages ('000 US$) scaled" (factor 1000 is applied) impact_instance.eai_exp holds expected impact for each country (within 1/frequency_unit). impact_instance.coord_exp holds rough central coordinates for each country.
- **countries** (*list of str*) – ISO3-codes of countries in same order as in impact_instance.eai_exp

4.1.2 climada.entity package

climada.entity.disc_rates package

climada.entity.disc_rates.base module

class climada.entity.disc_rates.base.**DiscRates**(*years: ndarray | None = None, rates: ndarray | None = None*)

Bases: `object`

Defines discount rates and basic methods. Loads from files with format defined in FILE_EXT.

years

list of years

Type
np.array

rates

list of discount rates for each year (between 0 and 1)

Type
np.array

__init__(*years: ndarray | None = None, rates: ndarray | None = None*)

Fill discount rates with values and check consistency data

Parameters

- **years** (*numpy.ndarray(int)*) – Array of years. Default is numpy.array([]).
- **rates** (*numpy.ndarray(float)*) – Discount rates for each year in years. Default is numpy.array([]). Note: rates given in float, e.g., to set 1% rate use 0.01

clear()

Reinitialize attributes.

check()

Check attributes consistency.

Raises
ValueError –

select(*year_range*)

Select discount rates in given years.

Parameters

- **year_range** (*np.array(int)*) – continuous sequence of selected years.
- **Returns** (*climada.entity.DiscRates*) – The selected discrates in the year_range

append(*disc_rates*)

Check and append discount rates to current DiscRates. Overwrite discount rate if same year.

Parameters

disc_rates (*climada.entity.DiscRates*) – DiscRates instance to append

Raises

ValueError –

net_present_value(*ini_year*, *end_year*, *val_years*)

Compute net present value between present year and future year.

Parameters

- **ini_year** (*float*) – initial year
- **end_year** (*float*) – end year
- **val_years** (*np.array*) – cash flow at each year btw ini_year and end_year (both included)

Returns

net_present_value – net present value between present year and future year.

Return type

float

plot(*axis=None*, *figsize=(6, 8)*, ***kwargs*)

Plot discount rates per year.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple(int, int), optional*) – size of the figure. The default is (6,8)
- **kwargs** (*optional*) – keyword arguments passed to plotting function axis.plot

Returns

axis – axis handles of the plot

Return type

matplotlib.axes._subplots.AxesSubplot

classmethod from_mat(*file_name*, *var_names=None*)

Read MATLAB file generated with previous MATLAB CLIMADA version.

Parameters

- **file_name** (*str*) – filename including path and extension
- **description** (*str, optional*) – description of the data. The default is ''
- **var_names** (*dict, optional*) – name of the variables in the file. Default:

```
>>> DEF_VAR_MAT = {
...     'sup_field_name': 'entity',
...     'field_name': 'discount',
...     'var_name': {
...         'year': 'year',
...         'disc': 'discount_rate',
...     }
... }
```

Returns
The disc rates from matlab

Return type
climada.entity.DiscRates

read_mat (**args*, ***kwargs*)

This function is deprecated, use `DiscRates.from_mat` instead.

classmethod from_excel (*file_name*, *var_names=None*)

Read excel file following template and store variables.

Parameters

- **file_name** (*str*) – filename including path and extension
- **description** (*str, optional*) – description of the data. The default is ''
- **var_names** (*dict, optional*) – name of the variables in the file. The Default is

```
>>> DEF_VAR_EXCEL = {
...     'sheet_name': 'discount',
...     'col_name': {
...         'year': 'year',
...         'disc': 'discount_rate',
...     }
... }
```

Returns
The disc rates from excel

Return type
climada.entity.DiscRates

read_excel (**args*, ***kwargs*)

This function is deprecated, use DiscRates.from_excel instead.

write_excel (*file_name*, *var_names=None*)

Write excel file following template.

Parameters

- **file_name** (*str*) – filename including path and extension
- **var_names** (*dict, optional*) – name of the variables in the file. The Default is

```
>>> DEF_VAR_EXCEL = {
...     'sheet_name': 'discount',
...     'col_name': {
...         'year': 'year',
...         'disc': 'discount_rate',
...     }
... }
```

classmethod from_csv (*file_name*, *year_column='year'*, *disc_column='discount_rate'*, ***kwargs*)

Read DiscRate from a csv file following template and store variables.

Parameters

- **file_name** (*str*) – filename including path and extension

- **year_column** (*str, optional*) – name of the column that contains the years, Default: "year"
- **disc_column** (*str, optional*) – name of the column that contains the discount rates, Default: "discount_rate"
- ****kwargs** – any additional arguments, e.g., *sep*, *delimiter*, *head*, are forwarded to `pandas.read_csv`

Returns
The disc rates from the csv file

Return type
climada.entity.DiscRates

write_csv(*file_name*, *year_column='year'*, *disc_column='discount_rate'*, ***kwargs*)

Write DiscRate to a csv file following template and store variables.

Parameters

- **file_name** (*str*) – filename including path and extension
- **year_column** (*str, optional*) – name of the column that contains the years, Default: "year"
- **disc_column** (*str, optional*) – name of the column that contains the discount rates, Default: "discount_rate"
- ****kwargs** – any additional arguments, e.g., *sep*, *delimiter*, *head*, are forwarded to `pandas.read_csv`

climada.entity.exposures package

climada.entity.exposures.litpop package

climada.entity.exposures.litpop.gpw_population module

climada.entity.exposures.litpop.gpw_population.**load_gpw_pop_shape**(*geometry*, *reference_year*, *gpw_version*, *data_dir=PosixPath('/home/docs/climada/data')*, *layer=0*, *verbose=True*)

Read gridded population data from TIFF and crop to given shape(s).

Note: A (free) NASA Earthdata login is necessary to download the data. Data can be downloaded e.g. for gpw_version=11 and year 2015 from https://sedac.ciesin.columbia.edu/downloads/data/gpw-v4/ gpw-v4-population-count-rev11/gpw-v4-population-count-rev11_2015_30_sec_tif.zip

Parameters

- **geometry** (*shape(s) to crop data to in degree lon/lat.*) – for example shapely.geometry.(Multi)Polygon or shapefile.Shape from polygon(s) defined in a (country) shapefile.
- **reference_year** (*int*) – target year for data extraction
- **gpw_version** (*int*) – Version number of GPW population data, i.e. 11 for v4.11. The default is CONFIG.exposures.litpop.gpw_population.gpw_version.int()
- **data_dir** (*Path, optional*) – Path to data directory holding GPW data folders. The default is SYSTEM_DIR.
- **layer** (*int, optional*) – relevant data layer in input TIFF file to return. The default is 0 and should not be changed without understanding the different data layers in the given TIFF file.

- **verbose** (*bool, optional*) – Enable verbose logging about the used GPW version and reference year. Default: True.

Returns

- **pop_data** (*2D numpy array*) – contains extracted population count data per grid point in shape first dimension is lat, second dimension is lon.
- **meta** (*dict*) – contains meta data per array, including "transform" with meta data on coordinates.
- **global_transform** (*Affine instance*) – contains six numbers, providing transform info for global GWP grid. global_transform is required for resampling on a globally consistent grid

climada.entity.exposures.litpop.gpw_population.**get_gpw_file_path**(*gpw_version*, *reference_year*, *data_dir=None*, *verbose=True*)

Check available GPW population data versions and year closest to *reference_year* and return full path to TIFF file.

Parameters

- **gpw_version** (*int (optional)*) – Version number of GPW population data, i.e. 11 for v4.11.
- **reference_year** (*int (optional)*) – Data year is selected as close to reference_year as possible. The default is 2020.
- **data_dir** (*pathlib.Path (optional)*) – Absolute path where files are stored. Default: SYSTEM_DIR
- **verbose** (*bool, optional*) – Enable verbose logging about the used GPW version and reference year. Default: True.

Raises

FileExistsError –

Returns

pathlib.Path

Return type

path to input file with population data

climada.entity.exposures.litpop.gpw_population.**grid_aligned_with_gpw**(*reference_year*, *gpw_version*, *res_arcsec*, *data_dir=PosixPath('/home/docs/climada/d*

Defines a grid based on population metadata.

Parameters

- **reference_year** (*int*) – The reference year for population and nightlight data.
- **gpw_version** (*int*) – Version number of GPW population data.
- **res_arcsec** (*int or None*) – Desired resolution in arcseconds. If None, aligns to population grid.
- **data_dir** (*str*) – Path to input data directory.

Returns

grid – A dictionary containing grid metadata, following the raster grid specification.

Return type

dict

climada.entity.exposures.litpop.litpop module

climada.entity.exposures.litpop.litpop.**GPW_VERSION = 11**

Version of Gridded Population of the World (GPW) input data. Check for updates.

class climada.entity.exposures.litpop.litpop.**LitPop**(**args*, *meta=None*, *exponents=None*, *fin_mode=None*, *gpw_version=None*, ***kwargs*)

Bases: *Exposures*

Holds geopandas GeoDataFrame with metadata and columns (pd.Series) defined in Attributes of Exposures class. LitPop exposure values are disaggregated proportional to a combination of nightlight intensity (NASA) and Gridded Population data (SEDAC). Total asset values can be produced capital, population count, GDP, or non-financial wealth.

Calling sequence example: country_names = ['CHE', 'Austria'] exp = LitPop.from_countries(country_names) exp.plot()

exponents

Defining powers (m, n) with which lit (nightlights) and pop (gpw) go into Lit**m * Pop**n. The default is (1,1).

Type
tuple of two integers, optional

fin_mode

Socio-economic value to be used as an asset base that is disaggregated. The default is 'pc'.

Type
str, optional

gpw_version

Version number of GPW population data, e.g. 11 for v4.11. The default is defined in GPW_VERSION.

Type
int, optional

__init__(**args*, *meta=None*, *exponents=None*, *fin_mode=None*, *gpw_version=None*, ***kwargs*)

Parameters

- **data** (*dict, iterable, DataFrame, GeoDataFrame, ndarray*) – data of the initial DataFrame, see `pandas.DataFrame()`. Used to initialize values for "region_id", "category_id", "cover", "deductible", "value", "geometry", "impf_[hazard type]".
- **columns** (*Index or array, optional*) – Columns of the initial DataFrame, see `pandas.DataFrame()`. To be provided if *data* is an array
- **index** (*Index or array, optional*) – Columns of the initial DataFrame, see `pandas.DataFrame()`. can optionally be provided if *data* is an array or for defining a specific row index
- **dtype** (*dtype, optional*) – data type of the initial DataFrame, see `pandas.DataFrame()`. Can be used to assign specific data types to the columns in *data*
- **copy** (*bool, optional*) – Whether to make a copy of the input *data*, see `pandas.DataFrame()`. Default is False, i.e. by default *data* may be altered by the `Exposures` object.
- **geometry** (*array, optional*) – Geometry column, see `geopandas.GeoDataFrame()`. Must be provided if *lat* and *lon* are None and *data* has no "geometry" column.

- **crs** (*value, optional*) – Coordinate Reference System, see `geopandas.GeoDataFrame()`.
- **meta** (*dict, optional*) – Metadata dictionary. Default: {} (empty dictionary). May be used to provide any of *description*, *ref_year*, *value_unit* and *crs*
- **description** (*str, optional*) – Default: None
- **ref_year** (*int, optional*) – Reference Year. Defaults to the entry of the same name in *meta* or 2018.
- **value_unit** (*str, optional*) – Unit of the exposed value. Defaults to the entry of the same name in *meta* or 'USD'.
- **value** (*array, optional*) – Exposed value column. Must be provided if *data* has no "value" column
- **lat** (*array, optional*) – Latitude column. Can be provided together with *lon*, alternative to *geometry*
- **lon** (*array, optional*) – Longitude column. Can be provided together with *lat*, alternative to *geometry*

set_countries(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_countries instead.

classmethod from_countries(*countries*, *res_arcsec=30*, *exponents=(1, 1)*, *fin_mode='pc'*, *total_values=None*, *admin1_calc=False*, *reference_year=2018*, *gpw_version=11*, *data_dir=PosixPath('/home/docs/climada/data')*, *target_grid=None*)

Init new LitPop exposure object for a list of countries (admin 0). 'Lit' and 'pop' data are first resampled to the same grid before being used to estimate the exposure value. The data are regridded based on the 'res_arcsec' argument or a predefined 'target_grid'. The regridding process is performed using 'util.coords.align_raster_data' with bilinear resampling ('rasterio.warp.Resampling.bilinear').

Sets attributes *ref_year*, *crs*, *value*, *geometry*, *meta*, *value_unit*, *exponents*,`fin_mode`, *gpw_version*, and *admin1_calc*.

Parameters

- **countries** (*list with str or int*) – list containing country identifiers: iso3alpha (e.g. 'JPN'), iso3num (e.g. 92) or name (e.g. 'Togo')
- **res_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arcsec, this corresponds to roughly 1 km. - To use the original **GPW population grid**, set 'res_arcsec=30'. - To use the original **nightlight grid**, set 'res_arcsec=15'. if 'target_grid' is defined, this argument is not used.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (nightlights) and pop (gpw) go into LitPop. To get nightlights^3 without population count: (3, 0). To use population count alone: (0, 1). Default: (1, 1)
- **fin_mode** (*str, optional*) – Socio-economic value to be used as an asset base that is disaggregated to the grid points within the country:
  - 'pc': produced capital (Source: World Bank), incl. manufactured or built assets such as machinery, equipment, and physical structures *pc* is in constant 2014 USD.
  - 'pop': population count (source: GPW, same as gridded population). The unit is 'people'.
  - 'gdp': gross-domestic product (Source: World Bank) [USD]
  - 'income_group': gdp multiplied by country's income group+1 [USD]. Income groups are 1 (low) to 4 (high income).

- 'nfw': non-financial wealth (Source: Credit Suisse, of households only) [USD]
- 'tw': total wealth (Source: Credit Suisse, of households only) [USD]
- 'norm': normalized by country (no unit)
- 'none': LitPop per pixel is returned unchanged (no unit)

Default: 'pc'

- **total_values** (*list containing numerics, same length as countries, optional*) – Total values to be disaggregated to grid in each country. The default is None. If None, the total number is extracted from other sources depending on the value of fin_mode.
- **admin1_calc** (*boolean, optional*) – If True, distribute admin1-level GDP (if available). Default: False
- **reference_year** (*int, optional*) – Reference year. Default: CONFIG.exposures.def_ref_year.
- **gpw_version** (*int, optional*) – Version number of GPW population data. The default is GPW_VERSION
- **data_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM_DIR.
- **target_grid** (*dict or None, optional*) – A predefined grid onto which the exposure data should be mapped. Default is None, the function will automatically define a target grid based on the 'res_arcsec' parameters. Specifically, if 'res_arcsec' is 15, the grid will be aligned to the nightlight data, otherwise it will align to the GPW population grid. The dictionary should include: - "crs" : Coordinate reference system (e.g., "EPSG:4326") - "transform" : Affine transformation defining pixel locations - "width" : Number of columns in the grid - "height" : Number of rows in the grid

Raises

`ValueError` –

Returns

exp – LitPop instance with exposure for given countries

Return type

LitPop

`set_nightlight_intensity`(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_nightlight_intensity instead.

`classmethod from_nightlight_intensity`(*countries=None*, *shape=None*, *res_arcsec=15*, *reference_year=2018*, *data_dir=PosixPath('/home/docs/climada/data')*, *target_grid=None*)

Wrapper around *from_countries* / *from_shape*.

Initiate exposures instance with value equal to the original BlackMarble nightlight intensity resampled based on the 'res_arcsec' argument or a predefined 'target_grid'. The regridding process is performed using 'util.coords.align_raster_data' with bilinear resampling ('rasterio.warp.Resampling.bilinear').

Provide either *countries* or *shape*.

Parameters

- **countries** (*list or str, optional*) – list containing country identifiers (name or iso3)
- **shape** (*Shape, Polygon or MultiPolygon, optional*) – geographical shape of target region, alternative to *countries*.

- **res_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arc-sec, this corresponds to roughly 1 km. - To use the original **GPW population grid**, set 'res_arcsec=30'. - To use the original **nightlight grid**, set 'res_arcsec=15'. if 'target_grid' is defined, this argument is not used.
- **reference_year** (*int, optional*) – Reference year. The default is CONFIG.exposures.def_ref_year.
- **data_dir** (*Path, optional*) – data directory. The default is None.
- **target_grid** (*dict or None, optional*) – A predefined grid onto which the exposure data should be mapped. Default is None, the function will automatically define a target grid based on the 'res_arcsec' parameters. Specifically, if 'res_arcsec' is 15, the grid will be aligned to the nightlight data, otherwise it will align to the GPW population grid. The dictionary should include: - "crs" : Coordinate reference system (e.g., "EPSG:4326") - "transform" : Affine transformation defining pixel locations - "width" : Number of columns in the grid - "height" : Number of rows in the grid

Raises

`ValueError` –

Returns

exp – Exposure instance with values representing pure nightlight intensity from input nightlight data (BlackMarble)

Return type

LitPop

`set_population`(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_population instead.

`classmethod from_population`(*countries=None*, *shape=None*, *res_arcsec=30*, *reference_year=2018*, *gpw_version=11*, *data_dir=PosixPath('/home/docs/climada/data')*, *target_grid=None*)

Wrapper around *from_countries* / *from_shape*.

Initiate exposures instance with value equal to GPW population count resampled to 'res_arcsec' or 'target_grid'. The regridding process is performed using 'util.coords.align_raster_data' with bilinear resampling ('rasterio.warp.Resampling.bilinear'). Provide either *countries* or *shape*.

Parameters

- **countries** (*list or str, optional*) – list containing country identifiers (name or iso3)
- **shape** (*Shape, Polygon or MultiPolygon, optional*) – geographical shape of target region, alternative to *countries*.
- **res_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arc-sec, this corresponds to roughly 1 km. - To use the original **GPW population grid**, set 'res_arcsec=30'. - To use the original **nightlight grid**, set 'res_arcsec=15'. if 'target_grid' is defined, this argument is not used.
- **reference_year** (*int, optional*) – Reference year (closest available GPW data year is used) The default is CONFIG.exposures.def_ref_year.
- **gpw_version** (*int, optional*) – specify GPW data verison. The default is 11.
- **data_dir** (*Path, optional*) – data directory. The default is None. Either countries or shape is required.
- **target_grid** (*dict or None, optional*) – A predefined grid onto which the exposure data should be mapped. Default is None, the function will automatically define a target grid based on the

'res_arcsec' parameters. Specifically, if 'res_arcsec' is 15, the grid will be aligned to the nightlight data, otherwise it will align to the GPW population grid. The dictionary should include: - "crs" : Coordinate reference system (e.g., "EPSG:4326") - "transform" : Affine transformation defining pixel locations - "width" : Number of columns in the grid - "height" : Number of rows in the grid

Raises

`ValueError` –

Returns

exp – Exposure instance with values representing population count according to Gridded Population of the World (GPW) input data set.

Return type

LitPop

`set_custom_shape_from_countries`(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_shape_and_countries instead.

`classmethod from_shape_and_countries`(*shape*, *countries*, *res_arcsec=30*, *exponents=(1, 1)*, *fin_mode='pc'*, *admin1_calc=False*, *reference_year=2018*, *gpw_version=11*, *data_dir=PosixPath('/home/docs/climada/data')*, *target_grid=None*)

create LitPop exposure for *country* and then crop to given shape. 'Lit' and 'pop' data are first resampled to the same grid before being used to estimate the exposure value. The data are regridded based on the 'res_arcsec' argument or a predefined 'target_grid'. The regridding process is performed using 'util.coords.align_raster_data' with bilinear resampling ('rasterio.warp.Resampling.bilinear').

Parameters

- **shape** (*shapely.geometry.Polygon, MultiPolygon, shapereader.Shape,*) – or GeoSeries or list containg either Polygons or Multipolygons. Geographical shape for which LitPop Exposure is to be initiated.
- **countries** (*list with str or int*) – list containing country identifiers: iso3alpha (e.g. 'JPN'), iso3num (e.g. 92) or name (e.g. 'Togo')
- **res_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arcsec, this corresponds to roughly 1 km. - To use the original **GPW population grid**, set 'res_arcsec=30'. - To use the original **nightlight grid**, set 'res_arcsec=15'. if 'target_grid' is defined, this argument is not used.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (nightlights) and pop (gpw) go into LitPop. Default: (1, 1)
- **fin_mode** (*str, optional*) – Socio-economic value to be used as an asset base that is disaggregated to the grid points within the country:
  - 'pc': produced capital (Source: World Bank), incl. manufactured or built assets such as machinery, equipment, and physical structures (pc is in constant 2014 USD)
  - 'pop': population count (source: GPW, same as gridded population). The unit is 'people'.
  - 'gdp': gross-domestic product (Source: World Bank) [USD]
  - 'income_group': gdp multiplied by country's income group+1 [USD] Income groups are 1 (low) to 4 (high income).
  - 'nfw': non-financial wealth (Source: Credit Suisse, of households only) [USD]
  - 'tw': total wealth (Source: Credit Suisse, of households only) [USD]

- 'norm': normalized by country
- 'none': LitPop per pixel is returned unchanged

Default: 'pc'

- **admin1_calc** (*boolean, optional*) – If True, distribute admin1-level GDP (if available). Default: False
- **reference_year** (*int, optional*) – Reference year for data sources. Default: 2020
- **gpw_version** (*int, optional*) – Version number of GPW population data. The default is GPW_VERSION
- **data_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM_DIR.
- **target_grid** (*dict or None, optional*) – A predefined grid onto which the exposure data should be mapped. Default is None, the function will automatically define a target grid based on the 'res_arcsec' parameters. Specifically, if 'res_arcsec' is 15, the grid will be aligned to the nightlight data, otherwise it will align to the GPW population grid. The dictionary should include: - "crs" : Coordinate reference system (e.g., "EPSG:4326") - "transform" : Affine transformation defining pixel locations - "resolution" : Spatial resolution in degrees - "width" : Number of columns in the grid - "height" : Number of rows in the grid

Raises

`NotImplementedError` –

Returns

exp – The exposure LitPop within shape

Return type

LitPop

`set_custom_shape`(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_shape instead.

`classmethod from_shape`(*shape*, *total_value*, *res_arcsec=30*, *exponents=(1, 1)*, *value_unit='USD'*, *region_id=None*, *reference_year=2018*, *gpw_version=11*, *data_dir=PosixPath('/home/docs/climada/data')*, *target_grid=None*)

init LitPop exposure object for a custom shape. Requires user input regarding the total value to be disaggregated.

Sets attributes *ref_year*, *crs*, *value*, *geometry*, *meta*, *value_unit*, *exponents*,`fin_mode`, *gpw_version*, and *admin1_calc*.

This method can be used to initiated LitPop Exposure for sub-national regions such as states, districts, cantons, cities, … but shapes and total value need to be provided manually. If these required input parameters are not known / available, better initiate Exposure for entire country and extract shape afterwards. 'Lit' and 'pop' data are first resampled to the same grid before being used to estimate the exposure value. The data are regridded based on the 'res_arcsec' argument or a predefined 'target_grid'. The regridding process is performed using 'util.coords.align_raster_data' with bilinear resampling ('rasterio.warp.Resampling.bilinear').

Parameters

- **shape** (*shapely.geometry.Polygon or MultiPolygon or shapereader.Shape.*) – Geographical shape for which LitPop Exposure is to be initiated.
- **total_value** (*int, float or None type*) – Total value to be disaggregated to grid in shape. If None, no value is disaggregated.

- **res_arcsec** (*float, optional*) – Horizontal resolution in arc-sec. The default is 30 arc-sec, this corresponds to roughly 1 km. - To use the original **GPW population grid**, set 'res_arcsec=30'. - To use the original **nightlight grid**, set 'res_arcsec=15'. if 'target_grid' is defined, this argument is not used.
- **exponents** (*tuple of two integers, optional*) – Defining power with which lit (nightlights) and pop (gpw) go into LitPop.
- **value_unit** (*str*) – Unit of exposure values. The default is USD.
- **region_id** (*int, optional*) – The numeric ISO 3166 region associated with the shape. If set to a value, this single value will be set for every coordinate in the `GeoDataFrame` of the resulting `LitPop` instance. If `None` (default), the region ID for every coordinate will be determined automatically (at a slight computational cost).
- **reference_year** (*int, optional*) – Reference year for data sources. Default: CONFIG.exposures.def_ref_year
- **gpw_version** (*int, optional*) – Version number of GPW population data. The default is set in CONFIG.
- **data_dir** (*Path, optional*) – redefines path to input data directory. The default is SYSTEM_DIR.
- **target_grid** (*dict or None, optional*) – A predefined grid onto which the exposure data should be mapped. Default is None, the function will automatically define a target grid based on the 'res_arcsec' parameters. Specifically, if 'res_arcsec' is 15, the grid will be aligned to the nightlight data, otherwise it will align to the GPW population grid. The dictionary should include: - "crs" : Coordinate reference system (e.g., "EPSG:4326") - "transform" : Affine transformation defining pixel locations - "width" : Number of columns in the grid - "height" : Number of rows in the grid

Raises

- **`NotImplementedError`** –
- **`ValueError`** –
- **`TypeError`** –

Returns

exp – The exposure LitPop within shape

Return type

LitPop

`set_country`(**args*, ***kwargs*)

This function is deprecated, use LitPop.from_countries instead.

`climada.entity.exposures.litpop.litpop.`**`get_value_unit`**(*fin_mode*)

get *value_unit* depending on *fin_mode*

Parameters

fin_mode (*Socio-economic value to be used as an asset base*)

Returns

value_unit

Return type

str

climada.entity.exposures.litpop.litpop.**reproject_input_data**(*data_array_list*, *meta_list*, *target_grid*, *resampling=Resampling.bilinear*, *conserve=None*)

LitPop-sepcific wrapper around u_coord.align_raster_data.

Reprojects all arrays in data_arrays to a given resolution – all based on the population data grid.

Parameters

- **data_array_list** (*list or array of numpy arrays containing numbers*) – Data to be reprojected, i.e. list containing N (min. 1) 2D-arrays. The data with the reference grid used to align the global destination grid to should be first data_array_list[i_align], e.g., pop (GPW population data) for LitPop.
- **meta_list** (*list of dicts*) – meta data dictionaries of data arrays in same order as data_array_list. Required fields in each dict are 'dtype,', 'width', 'height', 'crs', 'transform'. Example:

```
>>> {
...     'driver': 'GTiff',
...     'dtype': 'float32',
...     'nodata': 0,
...     'width': 2702,
...     'height': 1939,
...     'count': 1,
...     'crs': CRS.from_epsg(4326),
...     'transform': Affine(30/3600, 0.0, -18.175,
...                         0.0, -30/3600, 43.8),
... }
```

  The meta data with the reference grid used to define the global destination grid should be first in the list, e.g., GPW population data for LitPop.
- **target_grid** (*dict*) – A dictionary containing grid metadata, following the raster grid specification.
- **resampling** (*resampling function, optional*) – The default is rasterio.warp.Resampling.bilinear
- **conserve** (*str, optional, either 'mean' or 'sum'*) – Conserve mean or sum of data? The default is None (no conservation).

Returns

- **data_array_list** (*list*) – contains reprojected data sets
- **meta_out** (*dict*) – contains meta data of new grid (same for all arrays)

climada.entity.exposures.litpop.litpop.**gridpoints_core_calc**(*data_arrays*, *offsets=None*, *exponents=None*, *total_val_rescale=None*)

Combines N dense numerical arrays by point-wise multipilcation and optionally rescales to new total value:

(1) An offset (1 number per array) is added to all elements in the corresponding data array in data_arrays (optional).

(2) Numbers in each array are taken to the power of the corresponding exponent (optional).

(3) Arrays are multiplied element-wise.

(4) if total_val_rescale is provided, results are normalized and re-scaled with total_val_rescale.

(5) One array with results is returned.

Parameters

- **data_arrays** (*list or array of numpy arrays containing numbers*) – Data to be combined, i.e. list containing N (min. 1) arrays of same shape.
- **total_val_rescale** (*float or int, optional*) – Total value for optional rescaling of resulting array. All values in result_array are skaled so that the sum is equal to total_val_rescale. The default (None) implies no rescaling.
- **offsets** (*list or array containing N numbers >= 0, optional*) – One numerical offset per array that is added (sum) to the corresponding array in data_arrays. The default (None) corresponds to np.zeros(N).
- **exponents** (*list or array containing N numbers >= 0, optional*) – One exponent per array used as power for the corresponding array. The default (None) corresponds to np.ones(N).

Raises

ValueError – If input lists don't have the same number of elements. Or: If arrays in data_arrays do not have the same shape.

Returns

Results from calculation described above.

Return type

np.array of same shape as arrays in data_arrays

climada.entity.exposures.litpop.nightlight module

climada.entity.exposures.litpop.nightlight.**NOAA_RESOLUTION_DEG = 0.008333333333333333**

NOAA nightlights coordinates resolution in degrees.

climada.entity.exposures.litpop.nightlight.**NASA_RESOLUTION_DEG = 0.004166666666666667**

NASA nightlights coordinates resolution in degrees.

climada.entity.exposures.litpop.nightlight.**NASA_TILE_SIZE = (21600, 21600)**

NASA nightlights tile resolution.

climada.entity.exposures.litpop.nightlight.**NOAA_BORDER = (-180, -65, 180, 75)**

NOAA nightlights border (min_lon, min_lat, max_lon, max_lat)

climada.entity.exposures.litpop.nightlight.**BM_FILENAMES = ['BlackMarble_%i_A1_geo_gray.tif', 'BlackMarble_%i_A2_geo_gray.tif', 'BlackMarble_%i_B1_geo_gray.tif', 'BlackMarble_%i_B2_geo_gray.tif', 'BlackMarble_%i_C1_geo_gray.tif', 'BlackMarble_%i_C2_geo_gray.tif', 'BlackMarble_%i_D1_geo_gray.tif', 'BlackMarble_%i_D2_geo_gray.tif']**

Nightlight NASA files which generate the whole earth when put together.

climada.entity.exposures.litpop.nightlight.**BM_NIGHTLIGHT_GRID = {'crs': CRS.from_epsg(4326), 'driver': 'GTiff', 'dtype': 'float32', 'height': 43200, 'nodata': None, 'transform': (0.004166666666666667, 0.0, -180.0, 0.0, -0.004166666666666667, 90.0, 0.0, 0.0, 1.0), 'width': 86400}**

Grid aligned with Nightlight NASA files

climada.entity.exposures.litpop.nightlight.**load_nasa_nl_shape**(*geometry*, *year*, *data_dir=PosixPath('/home/docs/climada/data')*, *dtype='float32'*)

Read nightlight data from NASA BlackMarble tiles cropped to given shape(s) and combine arrays from each tile.

1) check and download required blackmarble files

2) read and crop data from each file required in a bounding box around the given *geometry*.
3) combine data from all input files into one array. this array then contains all data in the geographic bounding box around *geometry*.
4) return array with nightlight data

Parameters

- **geometry** (*shape(s) to crop data to in degree lon/lat.*) – for example shapely.geometry.(Multi)Polygon or shapefile.Shape. from polygon defined in a shapefile. The object should have attribute 'bounds' or 'points'
- **year** (*int*) – target year for nightlight data, e.g. 2016. Closest availble year is selected.
- **data_dir** (*Path (optional)*) – Path to directory with BlackMarble data. The default is SYSTEM_DIR.
- **dtype** (*dtype*) – data type for output default 'float32', required for LitPop, choose 'int8' for integer.

Returns

- **results_array** (*numpy array*) – extracted and combined nightlight data for bounding box around shape
- **meta** (*dict*) – rasterio meta data for results_array

climada.entity.exposures.litpop.nightlight.**get_required_nl_files**(*bounds*)

Determines which of the satellite pictures are necessary for
a certain bounding box (e.g. country)

Parameters
bounds (*1x4 tuple*) – bounding box from shape (min_lon, min_lat, max_lon, max_lat).

Raises
ValueError – invalid *bounds*

Returns
req_files – Array indicating the required files for the current operation with a boolean value (1: file is required, 0: file is not required).

Return type
numpy array

climada.entity.exposures.litpop.nightlight.**check_nl_local_file_exists**(*required_files=None, check_path=PosixPath('/home/docs/climad year=2016*)

Checks if BM Satellite files are avaialbe and returns a vector denoting the missing files.

Parameters

- **required_files** (*numpy array, optional*) – boolean array of dimension (8,) with which some files can be skipped. Only files with value 1 are checked, with value zero are skipped. The default is np.ones(len(BM_FILENAMES),)
- **check_path** (*str or Path*) – absolute path where files are stored. Default: SYSTEM_DIR
- **year** (*int*) – year of the image, e.g. 2016

Returns
files_exist – Boolean array that denotes if the required files exist.

Return type
numpy array

climada.entity.exposures.litpop.nightlight.**download_nl_files**(*req_files=array([1., 1., 1., 1., 1., 1., 1., 1.]), files_exist=array([0., 0., 0., 0., 0., 0., 0., 0.]), dwnl_path=PosixPath('/home/docs/climada/data'), year=2016*)

Attempts to download nightlight files from NASA webpage.

Parameters

- **req_files** (*numpy array, optional*) –

  Boolean array which indicates the files required (0-> skip, 1-> download).
  The default is np.ones(len(BM_FILENAMES),).

- **files_exist** (*numpy array, optional*) –

  Boolean array which indicates if the files already
  exist locally and should not be downloaded (0-> download, 1-> skip). The default is np.zeros(len(BM_FILENAMES),).

- **dwnl_path** (*str or path, optional*) – Download directory path. The default is SYSTEM_DIR.
- **year** (*int, optional*) – Data year to be downloaded. The default is 2016.

Raises

- **ValueError** –
- **RuntimeError** –

Returns
dwnl_path – Download directory path.

Return type
str or path

climada.entity.exposures.litpop.nightlight.**load_nasa_nl_shape_single_tile**(*geometry, path, layer=0*)

Read nightlight data from single NASA BlackMarble tile and crop to given shape.

Parameters

- **geometry** (*shape or geometry object*) – shape(s) to crop data to in degree lon/lat. for example shapely.geometry.Polygon object or from polygon defined in a shapefile.
- **path** (*Path or str*) – full path to BlackMarble tif (including filename)
- **layer** (*int, optional*) – TIFF-layer to be returned. The default is 0. BlackMarble usually comes with 3 layers.

Returns

- **out_image[layer,** (*,:] : 2D numpy ndarray*) – 2d array with data cropped to bounding box of shape
- **meta** (*dict*) – rasterio meta

climada.entity.exposures.litpop.nightlight.**load_nightlight_nasa**(*bounds*, *req_files*, *year*)

Get nightlight from NASA repository that contain input boundary.

Note: Legacy for BlackMarble, not required for litpop module

Parameters

- **bounds** (*tuple*) – min_lon, min_lat, max_lon, max_lat
- **req_files** (*np.array*) – array with flags for NASA files needed
- **year** (*int*) – nightlight year

Returns

- **nightlight** (*sparse.csr_matrix*)
- **coord_nl** (*np.array*)

climada.entity.exposures.litpop.nightlight.**read_bm_file**(*bm_path*, *filename*)

Reads a single NASA BlackMarble GeoTiff and returns the data. Run all required checks first.

Note: Legacy for BlackMarble, not required for litpop module

Parameters

- **bm_path** (*str*) – absolute path where files are stored.
- **filename** (*str*) – filename of the file to be read.

Returns

- **arr1** (*array*) – Raw BM data
- **curr_file** (*gdal GeoTiff File*) – Additional info from which coordinates can be calculated.

climada.entity.exposures.litpop.nightlight.**unzip_tif_to_py**(*file_gz*)

Unzip image file, read it, flip the x axis, save values as pickle and remove tif.

Parameters

file_gz (*str*) – file fith .gz format to unzip

Returns

- **fname** (*str*) – file_name of unzipped file
- **nightlight** (*sparse.csr_matrix*)

climada.entity.exposures.litpop.nightlight.**untar_noaa_stable_nightlight**(*f_tar_ini*)

Move input tar file to SYSTEM_DIR and extract stable light file. Returns absolute path of stable light file in format tif.gz.

Parameters

f_tar_ini (*str*) – absolute path of file

Returns

f_tif_gz – path of stable light file

Return type

str

climada.entity.exposures.litpop.nightlight.**load_nightlight_noaa**(*ref_year=2013*, *sat_name=None*)

Get nightlight luminosites. Nightlight matrix, lat and lon ordered such that nightlight[1][0] corresponds to lat[1], lon[0] point (the image has been flipped).

Parameters

- **ref_year** (*int, optional*) – reference year. The default is 2013.
- **sat_name** (*str, optional*) – satellite provider (e.g. 'F10', 'F18', …)

Returns

- **nightlight** (*sparse.csr_matrix*)
- **coord_nl** (*np.array*)
- **fn_light** (*str*)

climada.entity.exposures.base module

class climada.entity.exposures.base.**Exposures**(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*, *geometry=None*, *crs=None*, *meta=None*, *description=None*, *ref_year=None*, *value_unit=None*, *value=None*, *lat=None*, *lon=None*)

Bases: `object`

geopandas GeoDataFrame with metadata and columns (pd.Series) defined in Attributes.

description

metadata - description of content and origin of the data

Type
str

ref_year

metadata - reference year

Type
int

value_unit

metadata - unit of the exposures values

Type
str

data

containing at least the columns 'geometry' and 'value' for locations and assets optionally more, a.o., 'region_id', 'category_id', columns for (hazard specific) assigned centroids and (hazard specific) impact funcitons.

Type
GeoDataFrame

vars_oblig = ['value', 'geometry']

Name of the variables needed to compute the impact.

vars_def = ['impf_', 'if_']

Name of variables that can be computed.

vars_opt = ['centr_', 'deductible', 'cover', 'category_id', 'region_id', 'geometry']

Name of the variables that aren't need to compute the impact.

property crs

Coordinate Reference System, refers to the crs attribute of the inherent GeoDataFrame

property gdf

Inherent GeoDataFrame

property latitude

Latitude array of exposures

property longitude

Longitude array of exposures

property geometry

Geometry array of exposures

property value

Geometry array of exposures

property region_id

Region id for each exposure

Return type
np.array of int

property category_id

Category id for each exposure

Return type
np.array

property cover

Cover value for each exposures

Return type
np.array of float

property deductible

Deductible value for each exposures

Return type
np.array of float

hazard_impf(*haz_type=''*)

Get impact functions for a given hazard

Parameters
haz_type (*str*) – hazard type, as in the hazard's.haz_type which is the HAZ_TYPE constant of the hazard's module

Returns
impact functions for the given hazard

Return type
np.array of int

hazard_centroids(*haz_type=''*)

Get centroids for a given hazard

Parameters
haz_type (*str*) – hazard type, as in the hazard's.haz_type which is the HAZ_TYPE constant of the hazard's module

Returns
centroids index for the given hazard

Return type
np.array of int

derive_raster()

Metadata dictionary, containing raster information, derived from the geometry

__init__(*data=None*, *index=None*, *columns=None*, *dtype=None*, *copy=False*, *geometry=None*, *crs=None*, *meta=None*, *description=None*, *ref_year=None*, *value_unit=None*, *value=None*, *lat=None*, *lon=None*)

Parameters

- **data** (*dict, iterable, DataFrame, GeoDataFrame, ndarray*) – data of the initial DataFrame, see `pandas.DataFrame()`. Used to initialize values for "region_id", "category_id", "cover", "deductible", "value", "geometry", "impf_[hazard type]".
- **columns** (*Index or array, optional*) – Columns of the initial DataFrame, see `pandas.DataFrame()`. To be provided if *data* is an array
- **index** (*Index or array, optional*) – Columns of the initial DataFrame, see `pandas.DataFrame()`. can optionally be provided if *data* is an array or for defining a specific row index
- **dtype** (*dtype, optional*) – data type of the initial DataFrame, see `pandas.DataFrame()`. Can be used to assign specific data types to the columns in *data*
- **copy** (*bool, optional*) – Whether to make a copy of the input *data*, see `pandas.DataFrame()`. Default is False, i.e. by default *data* may be altered by the `Exposures` object.
- **geometry** (*array, optional*) – Geometry column, see `geopandas.GeoDataFrame()`. Must be provided if *lat* and *lon* are None and *data* has no "geometry" column.
- **crs** (*value, optional*) – Coordinate Reference System, see `geopandas.GeoDataFrame()`.
- **meta** (*dict, optional*) – Metadata dictionary. Default: {} (empty dictionary). May be used to provide any of *description*, *ref_year*, *value_unit* and *crs*
- **description** (*str, optional*) – Default: None
- **ref_year** (*int, optional*) – Reference Year. Defaults to the entry of the same name in *meta* or 2018.
- **value_unit** (*str, optional*) – Unit of the exposed value. Defaults to the entry of the same name in *meta* or 'USD'.
- **value** (*array, optional*) – Exposed value column. Must be provided if *data* has no "value" column
- **lat** (*array, optional*) – Latitude column. Can be provided together with *lon*, alternative to *geometry*
- **lon** (*array, optional*) – Longitude column. Can be provided together with *lat*, alternative to *geometry*

check()

Check Exposures consistency.

Reports missing columns in log messages.

set_crs(*crs='EPSG:4326'*)

Set the Coordinate Reference System. If the epxosures GeoDataFrame has a 'geometry' column it will be updated too.

Parameters
crs (*object, optional*) – anything anything accepted by pyproj.CRS.from_user_input.

set_gdf (*gdf: GeoDataFrame*, *crs=None*)

Set the *gdf* GeoDataFrame and update the CRS

Parameters

- **gdf** (*GeoDataFrame*)
- **crs** (*object, optional,*) – anything anything accepted by pyproj.CRS.from_user_input, by default None, then *gdf.crs* applies or - if not set - the exposure's current crs

get_impf_column (*haz_type=''*)

Find the best matching column name in the exposures dataframe for a given hazard type,

Parameters
haz_type (*str or None*) – hazard type, as in the hazard's.haz_type which is the HAZ_TYPE constant of the hazard's module

Returns

a column name, the first of the following that is present in the exposures' dataframe:

- `impf_[haz_type]`
- `if_[haz_type]`
- `impf_`
- `if_`

Return type
str

Raises
ValueError – if none of the above is found in the dataframe.

assign_centroids (*hazard*, *distance='euclidean'*, *threshold=None*, *overwrite=True*)

Assign for each exposure coordinate the closest hazard coordinate. The Exposures `gdf` will be altered by this method. It will have an additional (or modified) column named `centr_[hazard.HAZ_TYPE]` after the call.

Uses the utility function `u_coord.match_centroids`. See there for details and parameters.

The value -1 is used for distances larger than `threshold` in point distances.

Parameters

- **hazard** (*Hazard*) – Hazard to match. The hazard centroids must be in the same coordinate reference system (crs) as the exposures.
- **distance** (*str, optional*) – Distance to use for matching exposures points to hazard centroids Possible values are "euclidean", "haversine" and "approx". Default: "euclidean"
- **threshold** (*float*) – Assignement threshold in units of the exposure's crs. If the distance to the nearest neighbor exceeds *threshold*, the index *-1* is assigned. Set *threshold* to 0, to disable nearest neighbor matching and enforce exact matching. Default: twice the highest resolution (smallest distance between any two nearest neighbours) of the hazard centroids
- **overwrite** (*bool*) – If True, overwrite centroids already present. If False, do not assign new centroids. Default is True.

See also

climada.util.coordinates.match_coordinates
method to associate centroids to exposure points

climada.util.coordinates.estimate_matching_threshold
method to calculate the default threshold

climada.util.coordinates.km_to_degree
method to approximately convert kilometers to degrees

climada.util.coordinates.degree_to_km
method to approximately convert degrees to kilometers

Notes

For coordinates in lat/lon coordinates distances in degrees differ from distances in meters on the Earth surface, in particular for higher latitude and distances larger than 100km. If more accuracy for degree coordinates is needed, please use 'haversine' distance metric, which however is slower.

Caution: nearest neighbourg matching can introduce serious artefacts such as:

- exposure and hazard centroids with shifted grids can lead

to systematically wrong assignements. - hazard centroids covering larger areas than exposures may lead to sub-optimal matching if the threshold is too large - projected crs often diverge at the anti-meridian and close points on either side will be at a large distance. For proper handling of the anti-meridian please use degree coordinates in EPSG:4326. This might be relevant for countries like the Fidji or the US that cross the anti-meridian.

Users are free to implement their own matching alrogithm and save the matching centroid index in the appropriate column `centr_[hazard.HAZ_TYPE]`.

set_geometry_points(*scheduler=None*)

obsolete and deprecated since climada 5.0

Deprecated since version Obsolete: method call. As of climada 5.0, geometry points are set during object initialization

set_lat_lon()

Set latitude and longitude attributes from geometry attribute.

Deprecated since version latitude: and longitude columns are no longer meaningful in Exposures` GeoDataFrames. They can be retrieved from Exposures.latitude and .longitude properties

set_from_raster(**args*, ***kwargs*)

This function is deprecated, use Exposures.from_raster instead.

Deprecated since version The: use of Exposures.set_from_raster is deprecated. Use Exposures.from_raster instead.

classmethod from_raster(*file_name*, *band=1*, *src_crs=None*, *window=None*, *geometry=None*, *dst_crs=None*, *transform=None*, *width=None*, *height=None*, *resampling=Resampling.nearest*, *attrs=None*)

Read raster data and set latitude, longitude, value and meta

Parameters

- **file_name** (*str*) – file name containing values

- **band** (*int, optional*) – bands to read (starting at 1)
- **src_crs** (*crs, optional*) – source CRS. Provide it if error without it.
- **window** (*rasterio.windows.Windows, optional*) – window where data is extracted
- **geometry** (*list of shapely.geometry, optional*) – consider pixels only within these shape
- **dst_crs** (*crs, optional*) – reproject to given crs
- **transform** (*rasterio.Affine*) – affine transformation to apply
- **width** (*float*) – number of lons for transform
- **height** (*float*) – number of lats for transform
- **resampling** (*rasterio.warp,.Resampling optional*) – resampling function used for reprojection to dst_crs
- **attrs** (*dict*) – dictionary of kwargs passed to the resulting Exposures.__init__

Return type
Exposures

plot_scatter(*mask=None, ignore_zero=False, pop_name=True, buffer=0.0, extend='neither', axis=None, figsize=(9, 13), adapt_fontsize=True, title=None, **kwargs*)

Plot exposures geometry's value sum scattered over Earth's map. The plot will we projected according to the current crs.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple, optional*) – figure size for plt.subplots
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **title** (*str, optional*) – a title for the plot. If not set *self.description* is used.
- **kwargs** (*optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_hexbin(*mask=None, ignore_zero=False, pop_name=True, buffer=0.0, extend='neither', axis=None, figsize=(9, 13), adapt_fontsize=True, title=None, **kwargs*)

Plot exposures geometry's value sum binned over Earth's map. An other function for the bins can be set through the key reduce_C_function. The plot will we projected according to the current crs.

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False

- **pop_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max'] Default is 'neither'.
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple*) – figure size for plt.subplots Default is (9, 13).
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **title** (*str, optional*) – a title for the plot. If not set *self.description* is used.
- **kwargs** (*optional*) – arguments for hexbin matplotlib function, e.g. *reduce_C_function=np.average*. Default is *reduce_C_function=np.sum*

Return type
cartopy.mpl.geoaxes.GeoAxesSubplot

plot_raster(*res=None*, *raster_res=None*, *save_tiff=None*, *raster_f=<function Exposures.<lambda>>*, *label='value (log10)'*, *scheduler=None*, *axis=None*, *figsize=(9, 13)*, *fill=True*, *adapt_fontsize=True*, ***kwargs*)

Generate raster from points geometry and plot it using log10 scale *np.log10((np.fmax(raster+1, 1)))*.

Parameters

- **res** (*float, optional*) – resolution of current data in units of latitude and longitude, approximated if not provided.
- **raster_res** (*float, optional*) – desired resolution of the raster
- **save_tiff** (*str, optional*) – file name to save the raster in tiff format, if provided
- **raster_f** (*lambda function*) – transformation to use to data. Default: log10 adding 1.
- **label** (*str*) – colorbar label
- **scheduler** (*str*) – used for dask map_partitions. "threads", "synchronous" or "processes"
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **figsize** (*tuple, optional*) – figure size for plt.subplots
- **fill** (*bool, optional*) – If false, the areas with no data will be plotted in white. If True, the areas with missing values are filled as 0s. The default is True.
- **adapt_fontsize** (*bool, optional*) – If set to true, the size of the fonts will be adapted to the size of the figure. Otherwise the default matplotlib font size is used. Default is True.
- **kwargs** (*optional*) – arguments for imshow matplotlib function

Return type
matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot

plot_basemap(*mask=None*, *ignore_zero=False*, *pop_name=True*, *buffer=0.0*, *extend='neither'*, *zoom=10*, *url={'attribution': '(C) OpenStreetMap contributors (C) CARTO', 'html_attribution': '© OpenStreetMap contributors © CARTO', 'max_zoom': 20, 'name': 'CartoDB.Positron', 'subdomains': 'abcd', 'url': 'https://{s}.basemaps.cartocdn.com/{variant}/{z}/{x}/{y}{r}.png', 'variant': 'light_all'}*, *axis=None*, ***kwargs*)

Scatter points over satellite image using contextily

Parameters

- **mask** (*np.array, optional*) – mask to apply to eai_exp plotted. Same size of the exposures, only the selected indexes will be plot.
- **ignore_zero** (*bool, optional*) – flag to indicate if zero and negative values are ignored in plot. Default: False
- **pop_name** (*bool, optional*) – add names of the populated places, by default True.
- **buffer** (*float, optional*) – border to add to coordinates. Default: 0.0.
- **extend** (*str, optional*) – extend border colorbar with arrows. ['neither' | 'both' | 'min' | 'max']
- **zoom** (*int, optional*) – zoom coefficient used in the satellite image
- **url** (*Any, optional*) – image source, e.g., `ctx.providers.OpenStreetMap.Mapnik`. Default: `ctx.providers.CartoDB.Positron`
- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for scatter matplotlib function, e.g. cmap='Greys'. Default: 'Wistia'

Return type
matplotlib.figure.Figure, cartopy.mpl.geoaxes.GeoAxesSubplot

write_hdf5(*file_name*)

Write data frame and metadata in hdf5 format

Parameters
file_name (*str*) – (path and) file name to write to.

read_hdf5(**args*, ***kwargs*)

This function is deprecated, use Exposures.from_hdf5 instead.

classmethod from_hdf5(*file_name*)

Read data frame and metadata in hdf5 format

Parameters

- **file_name** (*str*) – (path and) file name to read from.
- **additional_vars** (*list*) – list of additional variable names, other than the attributes of the Exposures class, whose values are to be read into the Exposures object class.

Return type
Exposures

read_mat(**args*, ***kwargs*)

This function is deprecated, use Exposures.from_mat instead.

classmethod from_mat(*file_name*, *var_names=None*)

Read MATLAB file and store variables in exposures.

Parameters

- **file_name** (*str*) – absolute path file
- **var_names** (*dict, optional*) – dictionary containing the name of the MATLAB variables. Default: DEF_VAR_MAT.

Return type
Exposures

to_crs(*crs=None*, *epsg=None*, *inplace=False*)

Wrapper of the `GeoDataFrame.to_crs()` method.

Transform geometries to a new coordinate reference system. Transform all geometries in a GeoSeries to a different coordinate reference system. The crs attribute on the current GeoSeries must be set. Either crs in string or dictionary form or an EPSG code may be specified for output. This method will transform all points in all objects. It has no notion or projecting entire geometries. All segments joining points are assumed to be lines in the current projection, not geodesics. Objects crossing the dateline (or other projection boundary) will have undesirable behavior.

Parameters

- **crs** (*dict or str*) – Output projection parameters as string or in dictionary form.
- **epsg** (*int*) – EPSG code specifying output projection.
- **inplace** (*bool, optional, default: False*) – Whether to return a new GeoDataFrame or do the transformation in place.

Return type
None if inplace is True else a transformed copy of the exposures object

plot(**args*, ***kwargs*)

Wrapper of the `GeoDataFrame.plot()` method

copy(*deep=True*)

Make a copy of this Exposures object.

Parameters
deep (bool) (*Make a deep copy, i.e. also copy data. Default True.*)

Return type
Exposures

write_raster(*file_name*, *value_name='value'*, *scheduler=None*)

Write value data into raster file with GeoTiff format

Parameters
file_name (*str*) – name output file in tif format

static concat(*exposures_list*)

Concatenates Exposures or DataFrame objectss to one Exposures object.

Parameters
exposures_list (*list of Exposures or DataFrames*) – The list must not be empty with the first item supposed to be an Exposures object.

Returns
with the metadata of the first item in the list and the dataframes concatenated.

Return type
Exposures

centroids_total_value(*hazard*)

Compute value of exposures close enough to be affected by hazard

Deprecated since version 3.3: This method will be removed in a future version. Use `affected_total_value()` instead.

This method computes the sum of the value of all exposures points for which a Hazard centroid is assigned.

Parameters
hazard (*Hazard*) – Hazard affecting Exposures

Returns
Sum of value of all exposures points for which a centroids is assigned

Return type
float

affected_total_value(*hazard:* Hazard, *threshold_affected: float = 0*, *overwrite_assigned_centroids: bool = True*)

Total value of the exposures that are affected by at least one hazard event (sum of value of all exposures points for which at least one event has intensity larger than the threshold).

Parameters

- **hazard** (*Hazard*) – Hazard affecting Exposures
- **threshold_affected** (*int or float*) – Hazard intensity threshold above which an exposures is considere affected. The default is 0.
- **overwrite_assigned_centroids** (*boolean*) – Assign centroids from the hazard to the exposures and overwrite existing ones. The default is True.

Returns
Sum of value of all exposures points for which a centroids is assigned and that have at least one event intensity above threshold.

Return type
float

See also

Exposures.assign_centroids
method to assign centroids.

Note

The fraction attribute of the hazard is ignored. Thus, for hazards with fraction defined the affected values will be overestimated.

climada.entity.exposures.base.**add_sea**(*exposures*, *sea_res*, *scheduler=None*)

Add sea to geometry's surroundings with given resolution. region_id set to -1 and other variables to 0.

Parameters

- **exposures** (*Exposures*) – the Exposures object without sea surroundings.
- **sea_res** (*tuple (float,float)*) – (sea_coast_km, sea_res_km), where first parameter is distance from coast to fill with water and second parameter is resolution between sea points
- **scheduler** (*str, optional*) – used for dask map_partitions. "threads", "synchronous" or "processes"

Return type
Exposures

climada.entity.exposures.base.**INDICATOR_IMPF = 'impf_'**

Name of the column containing the impact functions id of specified hazard

climada.entity.exposures.base.**INDICATOR_CENTR = 'centr_'**

Name of the column containing the centroids id of specified hazard

climada.entity.impact_funcs package

climada.entity.impact_funcs.base module

class climada.entity.impact_funcs.base.**ImpactFunc**(*haz_type: str = '', id: str | int = '', intensity: ndarray | None = None, mdd: ndarray | None = None, paa: ndarray | None = None, intensity_unit: str = '', name: str = ''*)

Bases: object

Contains the definition of one impact function.

haz_type

hazard type acronym (e.g. 'TC')

Type
str

id

id of the impact function. Exposures of the same type will refer to the same impact function id

Type
int or str

name

name of the ImpactFunc

Type
str

intensity_unit

unit of the intensity

Type
str

intensity

intensity values

Type
np.array

mdd

mean damage (impact) degree for each intensity (numbers in [0,1])

Type
np.array

paa

percentage of affected assets (exposures) for each intensity (numbers in [0,1])

Type
np.array

__init__(*haz_type: str = '', id: str | int = '', intensity: ndarray | None = None, mdd: ndarray | None = None, paa: ndarray | None = None, intensity_unit: str = '', name: str = ''*)

Initialization.

Parameters

- **haz_type** (*str, optional*) – Hazard type acronym (e.g. 'TC').
- **id** (*int or str, optional*) – id of the impact function. Exposures of the same type will refer to the same impact function id.
- **intensity** (*np.array, optional*) – Intensity values. Defaults to empty array.
- **mdd** (*np.array, optional*) – Mean damage (impact) degree for each intensity (numbers in [0,1]). Defaults to empty array.
- **paa** (*np.array, optional*) – Percentage of affected assets (exposures) for each intensity (numbers in [0,1]). Defaults to empty array.
- **intensity_unit** (*str, optional*) – Unit of the intensity.
- **name** (*str, optional*) – Name of the ImpactFunc.

calc_mdr(*inten: float | ndarray*) → ndarray

Interpolate impact function to a given intensity.

Parameters
inten (*float or np.array*) – intensity, the x-coordinate of the interpolated values.

Return type
np.array

plot(*axis=None*, ***kwargs*)

Plot the impact functions MDD, MDR and PAA in one graph, where MDR = PAA * MDD.

Parameters

- **axis** (*matplotlib.axes._subplots.AxesSubplot, optional*) – axis to use
- **kwargs** (*optional*) – arguments for plot matplotlib function, e.g. marker='x'

Return type
matplotlib.axes._subplots.AxesSubplot

check()

Check consistent instance data.

Raises
ValueError –

classmethod from_step_impf(*intensity: tuple[float, float, float]*, *haz_type: str*, *mdd: tuple[float, float] = (0, 1)*, *paa: tuple[float, float] = (1, 1)*, *impf_id: int = 1*, ***kwargs*)

Step function type impact function.

By default, the impact is 100% above the step. Useful for high resolution modelling.

Parameters

- **intensity** (*tuple(float, float, float)*) – tuple of 3-intensity numbers: (minimum, threshold, maximum)
- **haz_type** (*str*) – the reference string for the hazard (e.g., 'TC', 'RF', 'WS', …)
- **mdd** (*tuple(float, float)*) – (min, max) mdd values. The default is (0, 1)
- **paa** (*tuple(float, float)*) – (min, max) paa values. The default is (1, 1)
- **impf_id** (*int, optional, default=1*) – impact function id
- **kwargs** – keyword arguments passed to ImpactFunc()